LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

Universāla RDF datubāzu tīmekļa saskarne

BAKALAURA DARBS

Autors: **Juris Ubagovskis**
Stud. apl. Nr. ju08041
Darba vadītājs: Dr.sc.comp. Kārlis Čerāns

RĪGA 2012

ANOTĀCIJA

Šī darba mērķis ir aplūkot rīkus, kuri ļauj lietotājam veidot informatīvās sistēmas bez nepieciešamības to izstrādē iesaistīt nozares speciālistus un bez nepieciešamības pašam veikt programmēšanas darbu, kā arī novērtēt to priekšrocības un trūkumus. Otrs darba mērķis ir aprakstīt arhitektūru platformai, kuru darba autors izstrādā semantisko datubāzu projekta ietvaros un kura ļauj veidot šādas lietotnes.

Atslēgvārdi: semantiskais tīmeklis, informatīvā sistēma, RDF

ABSTRACT

The aim of the paper is to examine the browsers which allow a user to create informative systems without the necessity to involve the specialists of the exact field in the process of output and without the necessity to perform the work of programming, as well as to evaluate the advantages and disadvantages of it. The second aim of the paper is to describe the architecture of the platform, which the author develops within the competence of the project of semantic databases and which allows to create such application software.

Keywords: semantic web, informative system, RDF

SATURS

APZĪMĒJUMU SARAKSTS

Jēdziens / apzīmējums	Skaidrojums
RDF	Resource description framework – ietvars datu aprakstīšanai sematiskajā tīmeklī.
HTML	Hiperteksta iezīmēšanas valoda (HyperText Markup Language) ir iezīmēšanas valoda, kas ir izstrādāta tīmekļa lappušu un citas pārlūkprogrammā attēlojamas informācijas glabāšanai
AJAX	AJAX (Asynchronous JavaScript and XML) tīmekļa aplikāciju izstrādes tehnika, kas ļauj veidot dinamiskas klienta lietotnes.
URI	URI (Uniform Resource Identifier) ir identifikators, kurš tiek lietots lai semantiskajā tīmeklī aprakstītu resursus.
W3C	Starptautiska kopiena (World Wide Web Consortium), kas izstrādā standartus, lai nodrošinātu ilgtermiņa tīmekļa izaugsmi
DBPS	Datubāzu pārvaldības sistēma ir programma, kura ļauj lietotājam veidot un kontrolēt datubāzes.
Skripts	Skripts ir instrukciju kopa, kura tiek interpretēta ar citas programmas palīdzību.

IEVADS

Semantiskās tehnoloģijas pasaulē kļūst arvien populārākas, un to attīstība turpinās. Viena no svarīgākajām semantisko tehnoloģiju daļām ir RDF (Resource Description Framework). RDF ir formāts, kas apraksta informāciju semantiskajā tīmeklī. Informācijas glabāšana RDF formātā pieaug, tādēļ ir jādomā par dažādām tehnoloģijām, kas ļautu ar šiem datiem strādāt, piemēram, informatīvajām sistēmām, kas balstītas uz semantiskajām tehnoloģijām.

Klasiska informatīvā sistēma sastāv no relāciju datubāzes un servera daļas un klienta lietotnes. RDB ir tehnisks formāts ar tehniskiem tabulu nosaukumiem, primārajām un ārējām atslēgām un datubāzes shēmas izstrādāšana tāpat kā klienta lietotnes izstrāde ir jāuztic programmētājiem.

Tomēr jau eksistē tehnoloģijas, kuras ļauj izveidot informatīvās sistēmas bez programmētāju iejaukšanās. Šādā veidā iespējam ietaupīt gan laiku, gan naudu. Viens šāds risinājums relāciju datubāzēm ir aprakstīts šī darba otrajā nodaļā.

Radīt informatīvo sistēmu bez programmēšanas ar semantiskajām tehnoloģijām būtu vēl vienkāršāk, jo aprakstīt datubāzes shēmu ar ontoloģijas palīdzību ir daudz vienkāršāk nekā tas ir ar relāciju modeli. Tādējādi šajā gadījumā visu informatīvās sistēmas izveides procesu spētu paveikt praktiski jebkurš lietotājs.

Tāds arī ir šī darba mērķi – apskatīt kādas tehnoloģijas jau ir pieejama, lai varētu veidot informatīvās sistēmas bez programmēšanas, kā arī aprakstīt arhitektūru platformai, kas ļautu šādas informatīvās sistēmas veidot.

Darbs ir organizēts četrās nodaļās. Pirmajā nodaļā aprakstīts autora agrāk izstrādāts RDF datu pārlūks. Otrajā nodaļā aprakstīts universāls datubāzu pārlūks relāciju datubāzēm, trešajā nodaļā aprakstīts tīmekļa vietņu ģenerators, kurš balstīts uz semantiskajām tehnoloģijām, bet ceturtajā nodaļa aprakstīta arhitektūra platformai, kuru autors izstrādā Latvijas Universitātes Matemātikas un Informātikas institūtā.

1. Vienkāršs RDF datu pārlūks

Šajā nodaļā tiks aprakstīts RDF datu pārlūks, kuru darba autors izstrādāja 2010. gadā.

RDF datu pārlūka ideja balstās uz to, ka lietotājam tiek piedāvāta iepriekš definēta saskarne, ar kuras palīdzību lietotājs var pārlūkot patvaļīgas RDF datubāzes saturu uz tās shēmas (šajā gadījumā ontoloģijas) pamata. Pārlūks ļauj ne tikai aplūkot datus, bet arī pievienot, dzēst un rediģēt formās redzamo informāciju.

Datu pārlūks darbam ar RDF datiem ir svarīgs jo, atšķirībā no relāciju datubāzēm, kur jau sen ir pieejami dažādi rīki, kas ļauj strādāt ar datubāzes saturu, kā arī pašas DBPS ļauj pārlūkot tabulas datus bez SQL vaicājumu rakstīšanas, RDF pasaulē nav daudz piemērotu pārlūku, kuri piedāvā strādāt ar lokālu RDF datu avotu. Pārsvarā pieejamie pārlūki paredzēti datu meklēšanai semantiskajā tīmeklī, nevis datubāzes satura pārlūkošanai un rediģēšanai uz shēmas pamata.

Izstrādātais RDF datu pārlūks šādu funkcionalitāti nodrošina – ērtu un intuitīvi saprotamu saskarni, ar kuras palīdzību lietotājs var skatīt informāciju uz ontoloģijas pamata. Lai strādātu ar RDF datiem nav nepieciešamas zināšanas ne RDF, ne OWL, ne SPARQL.

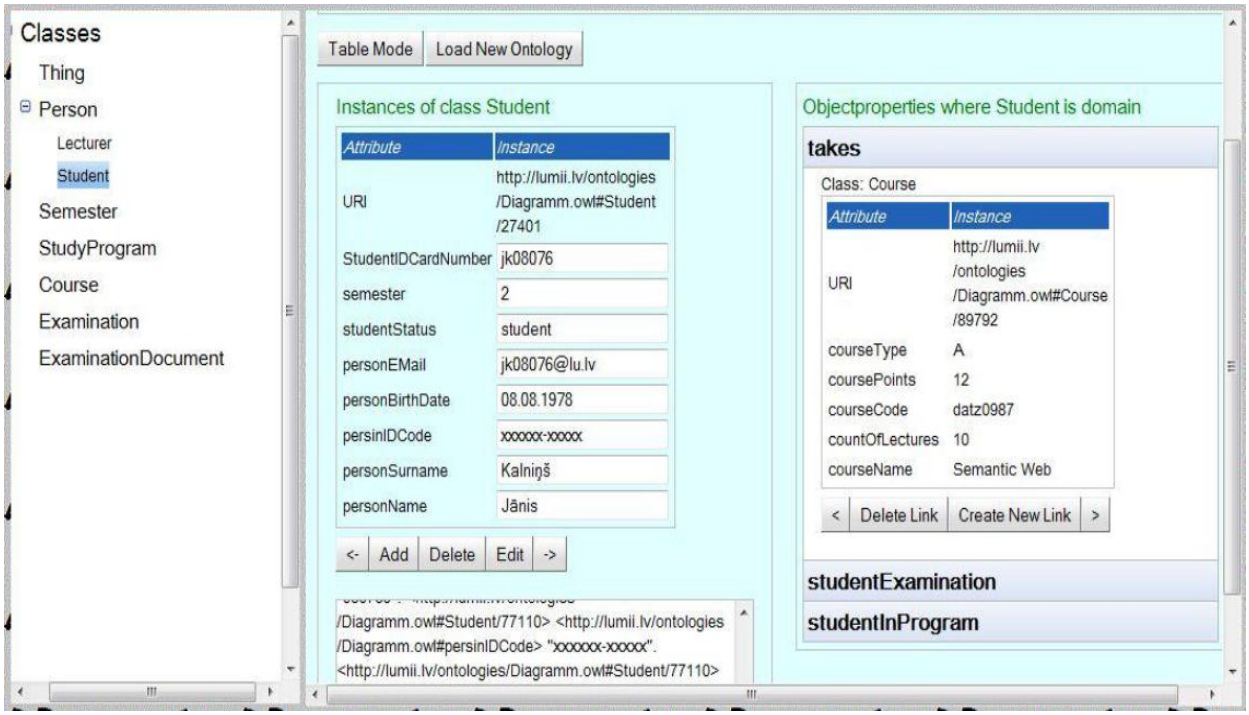
Papildus tiek nodrošināts tas, ka rediģējot datus nav iespējams radīt situāciju, ka dati vairs neatbilstu shēmai (ontoloģijai). Piemēram, ja ontoloģijā ir pateikts, ka klases students instances ir saistītas tikai ar klases Adrese instancēm, tad studenta instancei nevarēs patvaļīgi piekārtot citas klases instance. Otra lieta kas tiek pārbaudīta ir ievadīto datu tipu saderība ar atbilstošā ontoloģijas atribūta datu tipu, lai neatļautu, piemēram, personas vārda ievadlaukā glabāt skaitlisku vērtību.

Vēl viena pārlūka labā īpašība, kā tika minēts iepriekš, ir tāda, ka lai apskatītu datus nav nepieciešams rakstīt SPARQL vaicājumus, kuri ir grūti saprotami cilvēkiem, kuri ar tiem nav agrāk strādājuši. Tos automātiski ģenerē pats pārlūks. Vaicājumi tiek ģenerēti, par pamatu ņemot lietotāja ielādēto ontoloģiju.

Vēl tiek nodrošināts tas, ka šis RDF datu pārlūks spēj strādāt caur tīmekļa pārlūkprogrammu. Tādā veidā lietotājam var piekļūt datiem no jebkura datora.

1.1. Pārlūka dizains

Attēlā 1.1. ir attēlota pārlūka pamata saskarne:



1.1. att. **RDF pārlūka pamata saskarne**

Lapas kreisajā pusē tiek attēlots no ontoloģijas iegūtais klašu saraksts. Tas tiek veidots ņemot vērā virsklašu un apakšklašu attiecības, kā rezultāta saraksts tiek attēlots kokveida struktūrā.

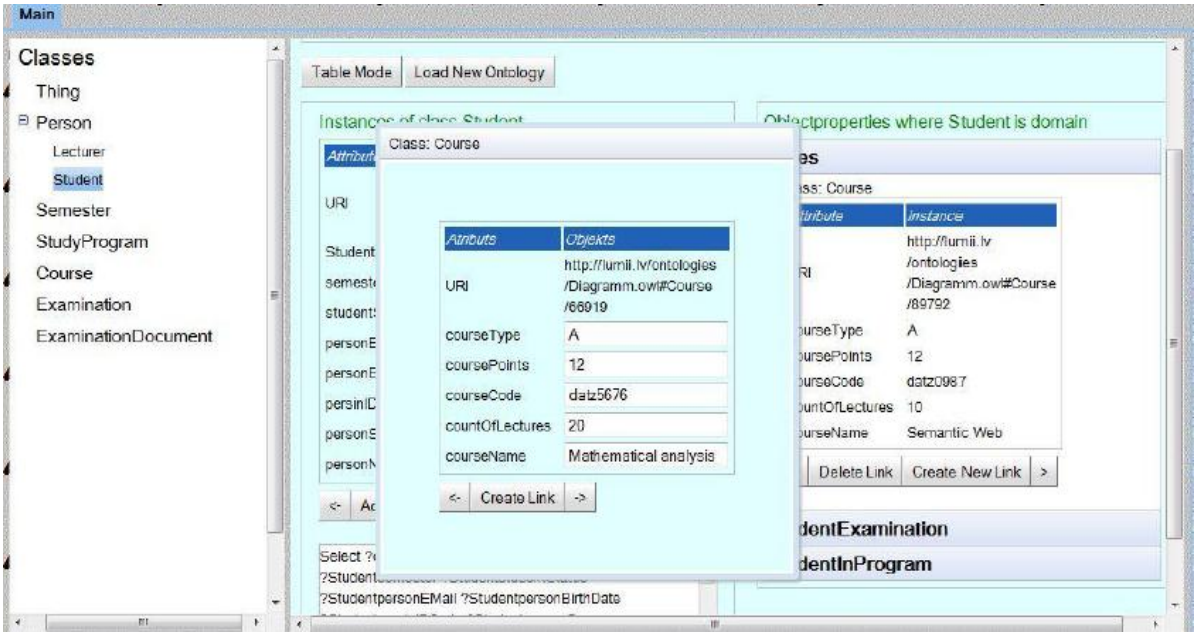
Lapas labā puse ir sadalīta divās daļās. Pa kreisi tiek attēlota instanču tabula (attēlā *Instances of class Student*). Tabulas pirmā kolona satur ontoloģijas izvēlētas klases atribūtu nosaukumus, bet otrā kolona – no datubāzes atlasītās instances datu vērtības, kuru saite atbilst atribūta nosaukumam. Zem tabulas atrodas navigācijas josla, kura ļauj pārvietoties pa atmiņā ielasītajām vērtībām. Vēl pieejamas pogas pievienot, dzēst un labot. Izvēloties pogu labot, tiks atvērts jauns dialoga logs, kurā lietotājs varēs ievadīt jaunās instances datu vērtības.

Izvēloties pogu dzēst, no datubāzes tiks izdzēsta gan pati instance, gan katrs trijnieks, kur kaut viena no apgalvojuma daļām saturēs dzēšamās instances URI.

Izvēloties pogu labot, tiks pārbaudītas datubāzē un tekstu laukos esošās vērtības. Ja kādas no tām nesaskanēs, jaunā vērtība tiks ierakstīta datubāzē.

Pa labi esošajā saišu panelī (attēlā *ObjectProperties where Student is domain*) ir izvietoti divi paneļi, kurās ir sadalītas izvēlētas klases sākuma gala asociācijas un beigu gala asociācijas.

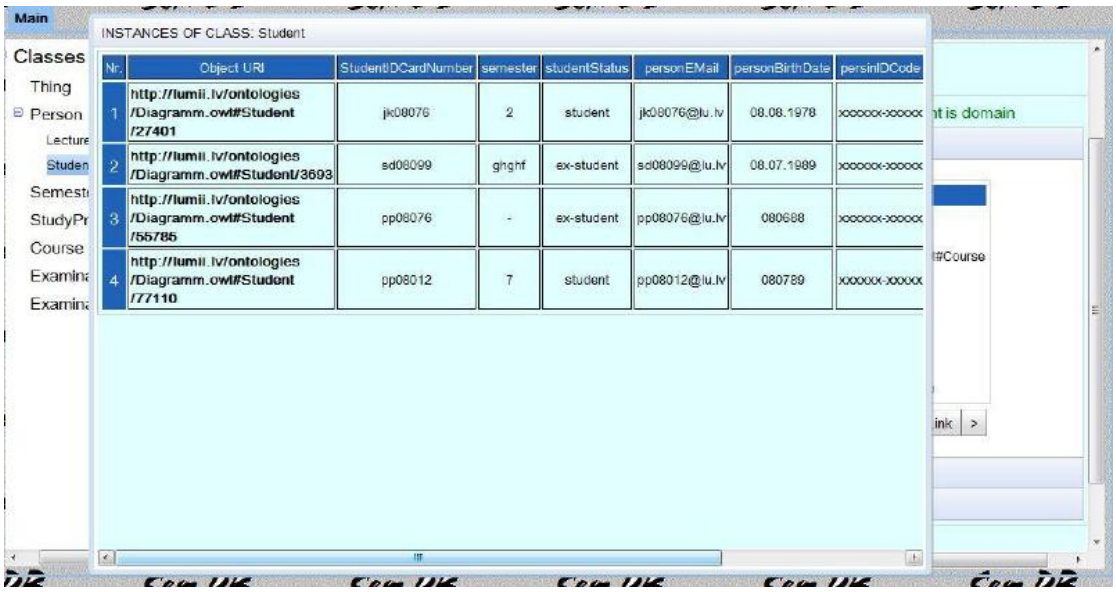
Zem katras asociācijas ir iznirstošā tabula, ar kuras palīdzību ir iespējams apskatīt izvēlētās klases saistītās instances. Tāpat ir iespējams veidot jaunas saites starp instancēm, izvēloties pogu *Izveidot saiti.* Tiks atvērts jauns dialoga logs ar tabulu, no kuras varēs izvēlēties kādu no instancēm, ar kuru saistīt izvēlēto instanci, attēls 1.2.



1.2. att. **Jaunas saites izveidošana**

Vēl pārlūks ļauj instances aplūkot ne tikai *braucot* tām cauri, bet arī tabulas veidā, attēls 1.3. Šāds skats ir noderīgs, ja klasei ir atlasītas daudzas instances. Tabulā izvēloties kādu no instancēm, tiks aizvērts tabulas režīms, un pārlūkā tiks rādīta izvēlētā instance tādā veidā, kā parādīts attēlā 1.1.

Lapas lejasdaļā tiek rādīts ģenerētais SPARQL pieprasījums, lai lietotājs tam varētu pievienot papildus daļas, piemēram, filtrus, jo šobrīd pārlūks to nenodrošina.



1.3.att. **Tabulas režīms**

1.2. Secinājumi

Izveidotais pārlūks ir noderīgs datubāzes datu aplūkošanai patvaļīgai RDF datubāzei, tomēr tā lielākais trūkums ir tas, ka informācija ir pārlūkojama tikai vienā iepriekš definētā formā. Lietotājs nevar ietekmēt to, kādā veidā informācija tiek attēlota un kāda informācija tiek attēlota. Šis pārlūks nav piemērots darbam ar lielām ontoloģijām. Piemēram ja klasei ir daudz atribūtu, formā tiks attēloti tie visi. Tādējādi saskarne vairs nav ērti lietojama.

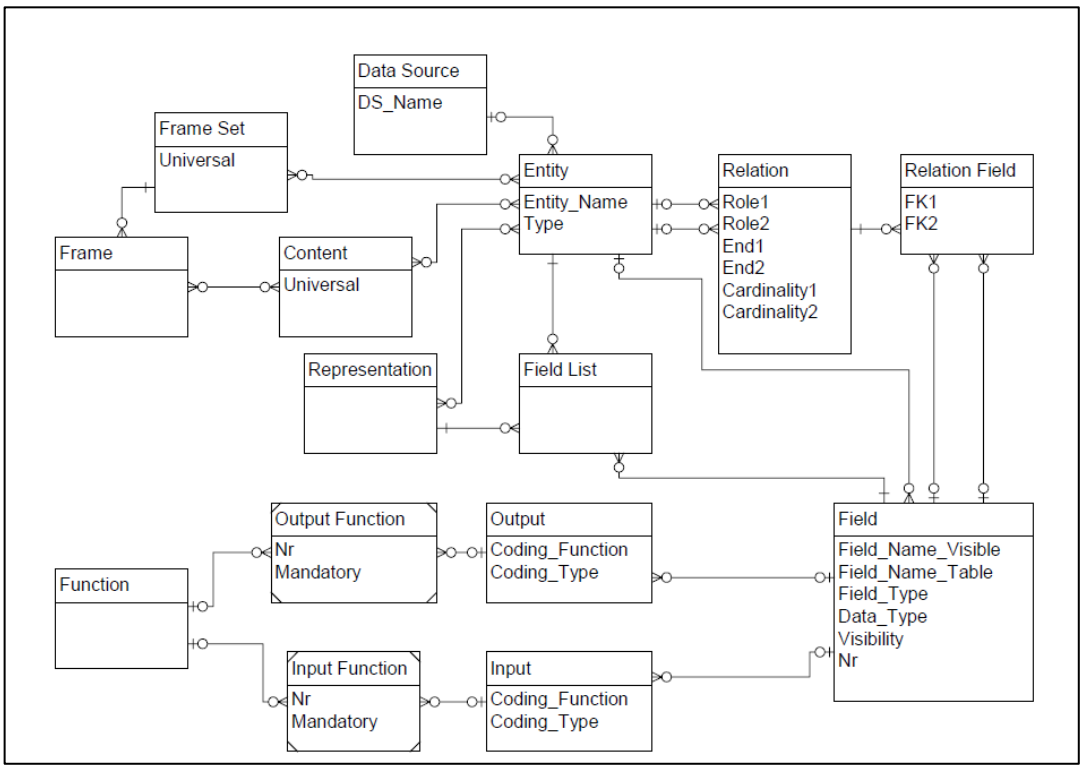
2. Datubāzu pārlūkošana uz datubāzes shēmas pamata

Šajā nodaļā tiks pastāstīts par universālu datubāzu pārlūku, kuru ir izveidojuši Latvijas Universitātes pasniedzēji Guntis Arnicāns un Ģirts Karnītis.

Universālā datubāzu pārlūka pamatā ir ideja par datu pārlūkošanu, kas ir balstīta uz datu (piemēram, ER(„entity-relationshiip")) modeli. Pārlūks piedāvā lietotājam strādā ar ģenerētu tīmekļa saskarni, kura tiek iepriekš definēta ar speciālas valodas palīdzību. Vēl viena šīs tehnoloģijas priekšrocība ir tāda, ka ar tās palīdzību ir iespējams strādāt ar datiem, kuri ir sadalīti starp vairākiem datu avotiem un ir veidoti dažādām tehnoloģijām. Tas tiek nodrošināts sadarbībai ar datu avotiem lietojot iesaiņotājus(angļu-val. *Wrapper).* [1]

2.1. Datu repozitorijs

Repozitorijs ir datubāze, kura satur informāciju par datu avotiem un saitēm starp tiem. Attēlā nr. 1.1. ir attēlots repozitorija konceptuālais ER modelis:



2.1. att. **Repozitorija metamodelis [1]**

Svarīgākās šī modeļa entītijas ir: [1]

- Entītijas *Data Source, Entity, Field, Relation, Relation Field* satur informāciju par datu avotu entītijām un relācijām.
- Entītijas *Function, Input, Input Function, Output, Output Function* satur informāciju par funkcijām, kas iegūst datus no datu avotiem, kā arī to ievada un izvada laukiem.
- Entītijas *Representation and Field List* satur informāciju par katras entītijas vizuālo reprezentāciju. Piemēram, kādus laukus rādīt, kādus nerādīt, kādā secībā rādīt.
- Entītijas *Frame Set, Frame and Content* satur informāciju par visas lapas vizuālo noformējumu.

2.2. Datu pārlūkošana

Universālā datubāzu pārlūka darbības principi balstās uz to, ka lapas tiek ģenerētas no lietotāja iepriekš definētām lapas definīcijām, kura tiek aizpildīta ar no datu avotiem iegūtajiem datiem. [1]

Tīmekļa lapa sastāv no rāmju(Frame) kopas(FrameSet). Rāmju kopā ir noteikts skaits rāmju un katram rāmim ir nodefinēts tā izmērs un novietojums lapā. Lietotājs var nodefinēt vairākas rāmju kopas, savukārt rāmju kopā var nodefinēt vairākus rāmjus. Šādi tiek nodrošināta organizēta informācijas apskate. Rāmju kopu var uzskatīt par skatu datiem, kas tiek iegūti no viena vai vairākiem datu avotiem. Katrā rāmju kopā ir viens galvenais rāmis un informācija pārējos šīs rāmju kopas rāmjos ir loģiski saistīta ar datiem galvenajā rāmī. Rāmji var saturēt vairākas vadīklas(angļu val. *Controls*), lai ļautu datiem būt sinhroniem visos rāmjos. [1]

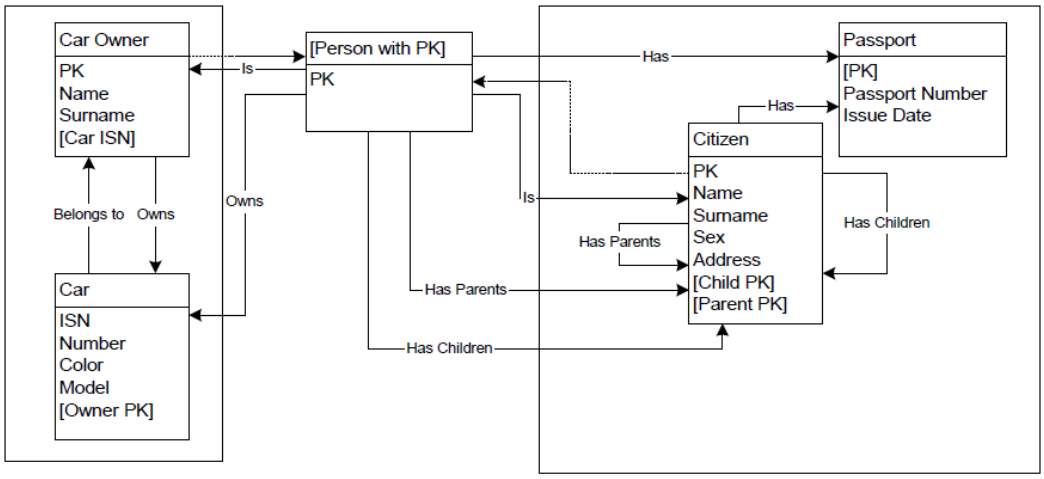
Katra rāmja izkārtojums (ang. - *layout*) ir definēts ar noteiktu likumu – Saturs (ang. – *Content),* kur saturs ir formula *Content(frameEntity, filterExpr). FrameEntity* ir kāda no entītijām datu avota metamodelī un *FilterExpression* ir loģiska izteiksme, kas filtrē datus no atbilstošā datu avota. Saturs definē izkārtojuma struktūru, kādi dati no metamodeļa un datu avotiem ir nepieciešami informācijas attēlošanai, kādas instances no definētajām entītijām ir saņemtas.

2.3. Tīmekļa lapu definēšana

Rāmju kopu un rāmju projektēšana balstās uz šablonu pielietošanas principu. Jaunas rāmju kopas un rāmjus paredzēts izstrādāt uz doto šablonu pamata. [1]

Tīmekļa lapas izstrādei ir divi galvenie posmi – rāmju kopas struktūras izplānošana un formulas izveide rāmja satura aizpildīšanai. [1]

Lai labāk izprastu šo procesu, tiks aprakstīts neliels piemērs. Sākumā paņemsim kādu datu avotu. Šajā gadījumā divas datubāzes – iedzīvotāju reģistru un transportlīdzekļu reģistru, attēls 1.2.



2.2. att. **Datu avotu ER modelis [1]**

Sāksim ar vienkāršu piemēru – nodefinēt tabulu, kurā tiek rādīta informācija par vienu personu (personas kods, vārds, uzvārds, dzimums, adrese), attēls 1.3.

PK	12121211111
Name	Andris
Surname	Kalns
Sex	M
Address	Riga, Liepu 1-12, LV-1000

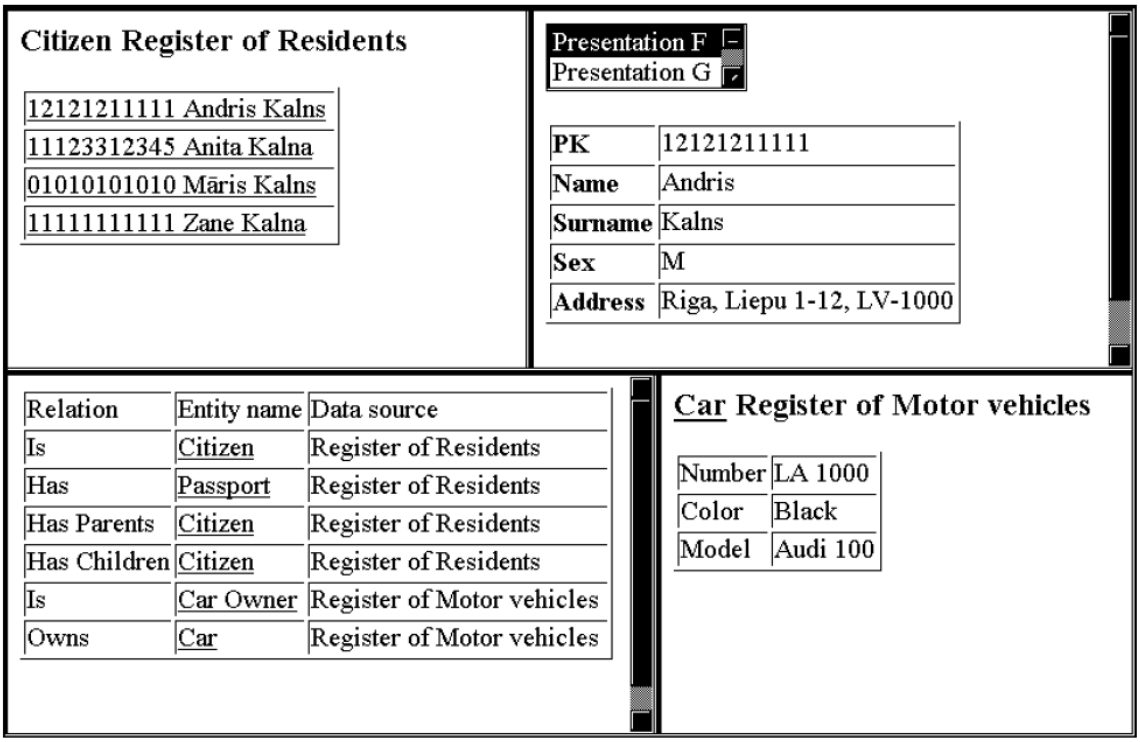
2.3. att. **Piemērs vienas instances attēlošanai [1]**

Definīcija šim piemēram:

- A(entity, record) = VerticalTable(List(A1, A2))
  A1 = IterateList(1%FieldList(entity, view), AO(SO(FieldName(1%))))
  A2 = IterateList(c(record, view), AO(SO(Value(2%))))

Pirmā kolona satur lauku nosaukumus, bet otrā šo laiku atbilstošās vērtības. Funkcija *VerticalTable* kā parametru saņem funkciju List, kura savukārt saņem divus parametrus A1 un A2. Funkcija List atgriež sarakstu no dotajiem elementiem, savukārt funkcija *VerticalTable* izveido *fObject* no sarakstu saraksta un šis freima objekts tiek attēlots kā tabula un iekšējie saraksti tiek izvietoti pa kolonām. *FObject* ir HTML instance, kas ir formatēta attēlošanai lapā. *IterateList* funkcija atgriež sarakstu, kas kā elementus satur rezultātus, kas iegūti pielietojot dotās funkcijas (šeit AO) katram saraksta elementam(1%FieldList, 1%FieldList). Funkcija AO, kas kā parametru saņem SO, pārveido uz *aObject* bez jebkādas aktivitātes. Attiecīgi *FieldName* un *Value* funkcijas, kas saņem parametrus, atgriež vērtības kā simbolu virknes.

Iepriekšējai piemērs parādīja kā nodefinēt vienkāršu tabulu. Tagad nodefinēsim vienu rāmju kopu. Par piemēru ņemsim attēlu 1.4.



2.4. att. **Rāmju kopas piemērs [1]**

Rāmju kopa sastāv no četriem rāmjiem. Pirmais rāmis attēlo iedzīvotājus no iedzīvotāju reģistra. Otrais rāmis attēlo informāciju, par pirmajā rāmī izvēlēto instanci. Trešais rāmis attēlo instances relācijas uz citām entītijā. Ceturtais rāmis attēlo pirmajā rāmī izvēlētās instances, caur trešajā rāmī izvēlēto relāciju, saistīto instanci. [1]

Definīcija pirmajam rāmim:

- I(entity, expr(entity)) = Vertical(I1, I2)
  I1 = Horizontal(List(FO(EntityName(entity)), FO(AO(SO(“ “))),
  FO(AO(SO(SourceName(entity))))))
  I2 = HorizontalTable(IterateList(7%RecordList,Link(B(7%), NULL, I3))
  I3 = List(Update(“FR_2”, entity, expr(entity) and expr(7%), “F”),
  Update(“FR_3”, entity, expr(entity) and expr(7%), “”), Clear(“FR_4”))

Definīcija otrajam rāmim:

- F(entity, expr(entity)) = Vertical(H, FO(AO(SO(“ “))), E5)
  G(entity, expr(entity)) = Vertical(H, FO(AO(SO(“ “))), Vertical(E5, G1))
  G1 = C(entity), where C3 is substitute with G2 in all places (we have added the action)
  G2 = FO(Link(SO(EntityName(RelationEntity(4%))), NULL, G3))
  G3 = List(Update(“FR_4”, RelationEntity(4%), expr(RelationEntity(4%)), “E”))
  H = ListBox(List(Link(“Presentation F”, NULL, H1), Link(“Presentation G”, NULL, H2)))
  H1 = Update(“FR_2”, entity, expr(entity), “F”)
  H2 = Update(“FR_2”, entity, expr(entity), “G”)

Definīcija trešajam rāmim:

- J(entity, expr(entity)) = D(entity, expr(entity))
  J1 = Link(SO(EntityName(RelationEntity(5%))), NULL, J2)
  J2 = List(Update(“FR_4”, RelationEntity(5%), expr(RelationEntity(5%)), “E”))

Definīcija ceturtajam rāmim:

- E(entity, expr(entity)) = Vertical(E1, E5)
  E1 = Horizontal(List(FO(E2), FO(AO(SO(“ “))), FO(AO(SO(SourceName(entity))))))
  E2 = Link(SO(EntityName(entity)), E3, E4)
  E3 = Navigate(“FRS_1”, entity, expr(entity), “”)
  E4 = List(Clear(“FR_2”), Update(“FR_3”, entity, expr(entity), “””), Clear(FR_4))
  E5 = Vertical(IteateList(6%RecordList(entity, expr(entity)), A(entity, 6%)))

2.4. Secinājumi

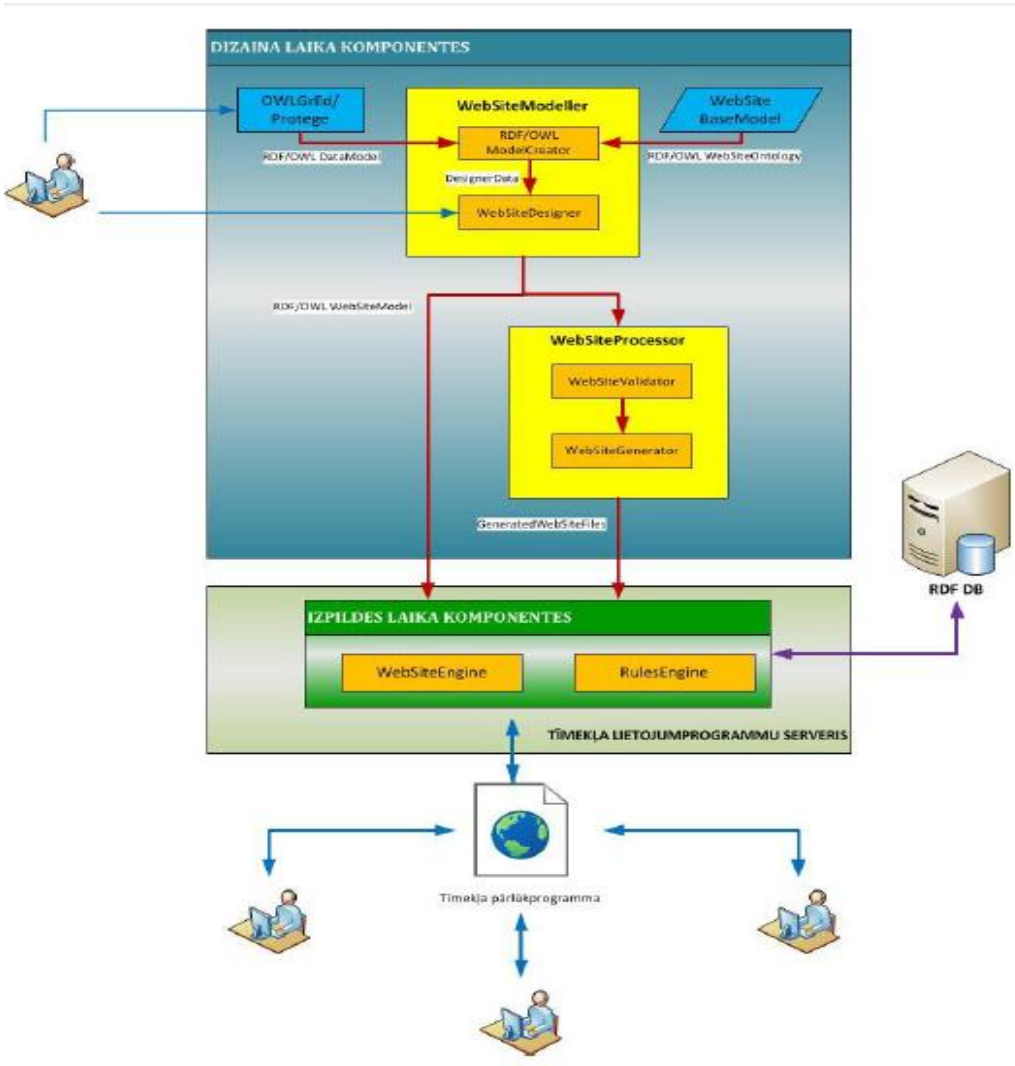
Šis risinājums ir interesants, jo ļauj veidot lietotnes, kuras var strādāt gan ar relāciju datubāzēm, gan arī grafu veidā aprakstītām struktūrām. Vēl viena sistēmas priekšrocība ir spēja veidot saskarni, kur dati tiek ņemti no dažādām datubāzēm. Šīs funkcionalitātes nodrošina iesaiņotāju sadarbībai ar datubāzēm.

Pēc autora domām sistēmas trūkums ir tas, ka informāciju paredzēts attēlot tikai tabulas veida struktūrās, kā arī sistēmas izpētes laikā radās iespaids, ka nav iespējams definēt sarežģītas struktūras, piemēram, kokus, kuru mezgli ir tabulas. Vēl kā neliels trūkumu varētu uzskatīt to, ka formas jādefinē ar valodas palīdzību, jo sanāk samērā neuzskatāmi.

3. Tīmekļa vietņu ģenerators

Šajā nodaļā tiks pastāstīts par rīku, kuru Sem DB projekta ietvaros ir izstrādājusi Aiga Romāne.

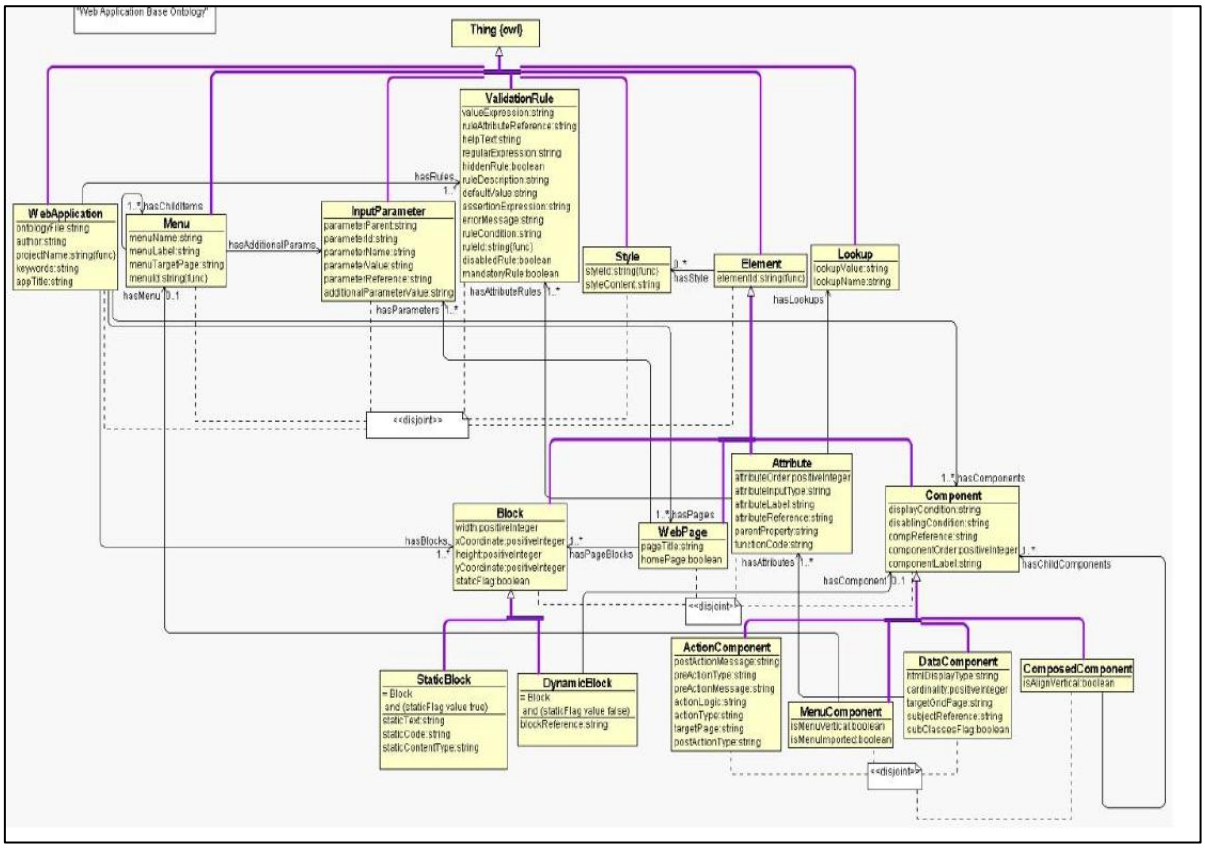
Šī rīka galvenā ideja balstās uz to, ka tā ļauj lietotājam izveidot tīmekļa vietni (informatīvo sistēmu) uz lietotāja izstrādātas datu ontoloģijas pamata. Sistēma sastāv no divām daļām – dizaina daļas un izpildes laika daļas. Dizaina laikā lietotājs izstrādā datu modeli, kuru pēc tam papildina ar tīmekļa vietnes metamodeļa klasēm un instancēm, savukārt izpildes laika daļā tiek ģenerēta tīmekļa vietne, ar kuru lietotājs var strādāt. [2]



3.1. att. **Risinājuma arhitektūra [2]**

3.1. Vietnes dizaina izstrāde

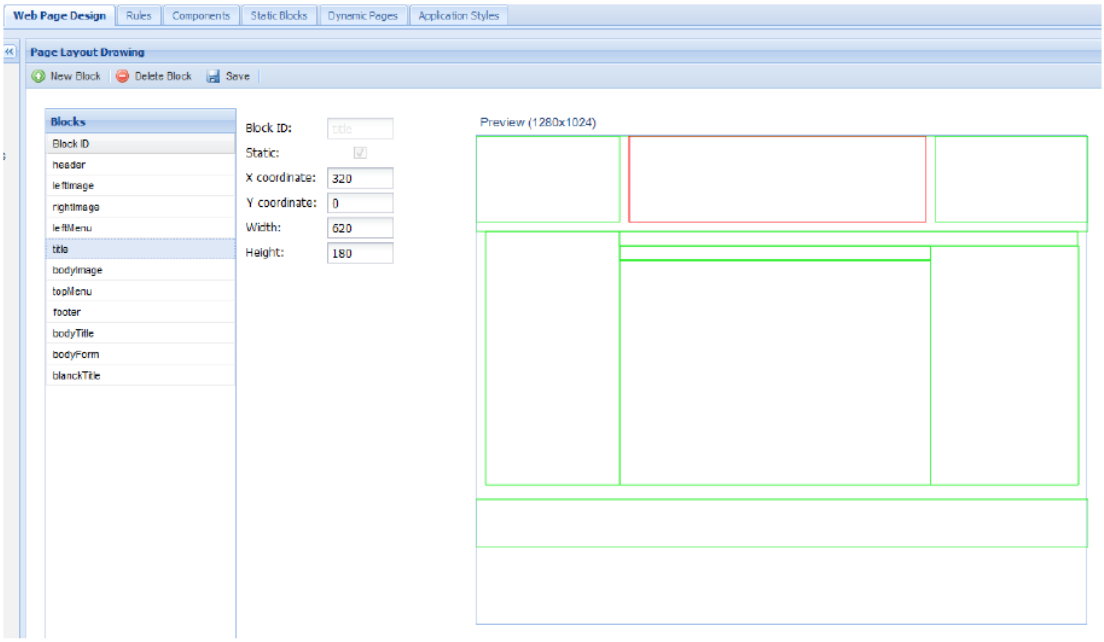
Dizaina daļā ieejā tiek saņemta lietotāja izstrādātā ontoloģiju (redzams attēlā 3.1), kurai ir jābūt RDF/XML faila formātā. Tālāk lietotāja izveidotā datu ontoloģija tiek sasaistīta ar bāzes tīmekļa vietnes ontoloģiju, kuras meta modelis ir attēlots attēlā 3.2.



3.2. att. **Tīmekļa vietnes bāzes ontoloģija [6]**

Attēlā redzams, ka tipiski tīmekļa vietne sastāv no tīmekļa lapām, kuras sastāv no blokiem. Katrs bloks ir vai nu statisks bloks, vai dinamisks bloks. Statiskā blokā var ievietot statiska veida informāciju, piemēram, tekstu vai attēlu. Dinamiskos blokā var ievietot dinamisku informāciju, piemēram, datus kas atlasīti no RDF datu grafa, vai datu ontoloģijas lauku nosaukumus. Dinamiskam bloka atbilst kāda no komponentēm – izvēlnes, datu, vai saliktā komponente. [2]

Kad lietotājs ir padevis datu ontoloģiju, un tā ir sasaistīta ar tīmekļa vietnes bāzes, tālāk darbs norit ar grafisku saskarni, kurā lietotājs definē tīmekļa vietnes struktūru (attēls 3.3.), un sasaista vietnes blokus ar definīciju, kura nosaka, kāda veida informācija blokā tiks rādīta.



3.3. att. **Tīmekļa vietnes konfigurācijas saskarne [6]**

Ar šīs saskarnes palīdzību lietotājs var nokonfigurēt lapas bloku izvietojumu, veidot datu ontoloģijas atribūtu validācijas likumus, definēt komponentes un piekārtot dažādām komponentēm stilus. [2]

Pēc tam, kad visa lietotnei nepieciešamā informācija ir ievadīta, tiek ģenerēta atbilstošās vietnes pilna ontoloģija, kas satur bāzes ontoloģijas klases, un tām piekārtotās instances, kuras atbilst informācijai, kura tika ievadīta ar konfigurācijas saskarnes palīdzību. Kā rezultāts tiek atgriezts ontoloģijas fails RDF/XML sintaksē, kurš pēc tam tiks lietots saskarnes ģenerēšanā. [2]

3.1. Izpildes laika daļa

Izpildes laika daļā tiek ģenerēta klienta – servera tipa tīmekļa lietotne. Lietotne tiek ģenerēta no ontoloģijas faila, kurš iegūts dizaina daļā. Iegūtais ontoloģijas fails satur visu nepieciešamo informāciju, lai varētu ģenerēt vietni, kura atbilst lietotāja definīcijai.

Izpildes laika daļa ir izstrādāta ar GWT/GXT tehnoloģiju, tāpēc ģenerēto tīmekļa lietojumprogrammu var lietot caur jebkuru tīmekļa pārlūkprogrammu. Ģeneratora programma kopā ar ontoloģijas datni jānovieto uz tīmekļa servera un tai var piekļūt no jebkuras vietas.

Datubāzes vaicājumiem tiek izmantota SPARQL valoda un paši vaicājumi tiek ģenerēti automātiski. Vaicājumu ģenerators saņem ieejas parametrus (domēnu vārdus, datu un objektu tipu īpašības, konkrētas īpašību vērtības, ja tādas ir, un arī parametrus, kas jāsameklē un jāizpilda ar datiem) un atgriež uzģenerētu pilnu vaicājumu. Pēc vaicājuma izpildes atgrieztie rezultāti tiek transformēti uz attiecīgās lapas datu modeli un tiek aizpildītas nepieciešamās komponentes ar datiem. [2]

Kā RDF datu glabātuve tiek izmantots Sesame API. Jaunākajā versijā ir pieejama arī Virtuoso datubāze.

3.2. Secinājumi

Tīmekļa vietņu izstrāde ir balstīta uz semantiskajām tehnoloģijām un ļauj viegli izstrādāt vienkāršas tīmekļa vietnes, kuras spēj sadarboties ar RDF datubāzi. To izstrādei ir nodrošināts ērts konfigurācijas interfeiss. Lapu un bloku definēšanai nav nepieciešamas speciālas iemaņas vai valodu zināšanas, kā iepriekš apskatītajā nodaļa.

Kā trūkumu var minēt, ka šeit netiek piedāvātas universālas formas, bet gan katrai ontoloģijas klasei specifisks datu attēlojums formā. Tāpat netiek atbalstīta datu piesaiste vairākos hierarhiski veidotos komponenšu struktūru līmeņos.

4. Universāla RDF datubāzu tīmekļa saskarne

Šajā nodaļā tiks aprakstīta arhitektūra platformai universālu RDF datubāzu tīmekļa saskarņu veidošanai.

Semantisko datubāzu projekta ietvaros tiek izstrādāta arhitektūra universālai konfigurējamu un dinamisku tīmekļa formu-lietotņu veidošanai ar mērķi to pielietot RDF datu pārlūkošanai un rediģēšanai. Viens no galvenajiem arhitektūras mērķiem ir nodrošināt lietotājam iespēju pašam konfigurēt saskarni darbam ar datiem, kā arī ļaut aprakstīt sarežģītus saskarnes elementus un attēlot dažādām struktūrām atbilstošus datus kā tabulas, kokus, sarakstus u.c.

Tādejādi ar šo platformu izveidotā tīmekļa lietotnes atbilstu informatīvās sistēmas jēdzienam. Informatīvai sistēmai jāatbilst trīs pamata noteikumiem.

- Informatīvajai sistēmai jābūt pieejamai lietotājam pastāvīgi un informācijas apmaiņai starp sistēmu un lietotāju jānotiek adekvātā laika periodā.
- Informācijai jābūt pareizai un uzticamai.
- Darbam ar informāciju ir jānodrošina ērta un viegli saprotama saskarne.

Ar šīs platformas palīdzību veidotās lietotnes atbilst visiem trīs noteikumiem. Lietotnes tiek veidotas kā tīmekļa lietotnes, kuras atrodas uz tīmekļa servera un pieejamība ir atkarība tikai no aparatūras kvalitātes.

Otrkārt, informācijas apmaiņa notiek ar RDF datubāzu serveri, kurš nodrošina datu glabāšanu un drošību. Datu pareizību nodrošinās pati lietotne – netiks ļauts izveidot situāciju, kad dati ir pretrunā art to, kas aprakstīts to shēmā.

Treškārt, saskarni varēs definēt pats sistēmas lietotājs, un tādejādi tā atbildīs katra paša priekšstatam par ērtu saskarni.

Tipiski informatīvās sistēmas sastāv no datubāzes (parasti RDB) un klienta lietotnes. Abas šīs komponentes tiek programmētas un tādejādi to izstrāde prasa laiku un naudas līdzekļus. Lietojot šo platformu, lietotājs pats varēs izveidot informatīvo sistēmu bez programmēšanas.

4.1. GWT

Tā kā platforma tiek izstrādāta ar GWT tehnoloģiju, šajā nodaļā par to tiks pastāstīts sīkāk.

GWT(Google Web Toolkit) ir atvērtā pirmkoda ietvars dinamisku tīmekļa lietotņu izstrādei. GWT tika izvēlēts tādēļ, ka tas ļauj tīmekļa lietotnes veidot ar valodas Java līdzekļiem, kas ir piemērota šādu projektu izstrādē, jo valoda Java tika radīta ar mērķi lielas un sarežģītas programmas padarīt viegli izstrādājamas, uzturamas un papildināmas. Vēl viens iemesls, kādēļ tika izvēlēts GWT, ir tas, ka izstrādes process norit, lietojot vienu valodu un ietvaru, jo parasti rakstot dinamiskas tīmekļa lietotnes, klienta puses daļa būtu jāizstrādā ar AJAX (Asynchronous JavaScript and XML) tehnoloģiju, bet servera puses kods, ar kādu citu valodu.

Vēl viena papildus priekšrocība ir tāda, ka strādājot ar GWT lietotājs var lietot populārās Java izstrādes vides(Eclipse, NetBeans) ar tajās iebūvētajiem atkļūdotājiem, kā arī citas valodai Java izstrādātās tehnoloģijas, piemēram, automātiskās testēšanas rīkus.

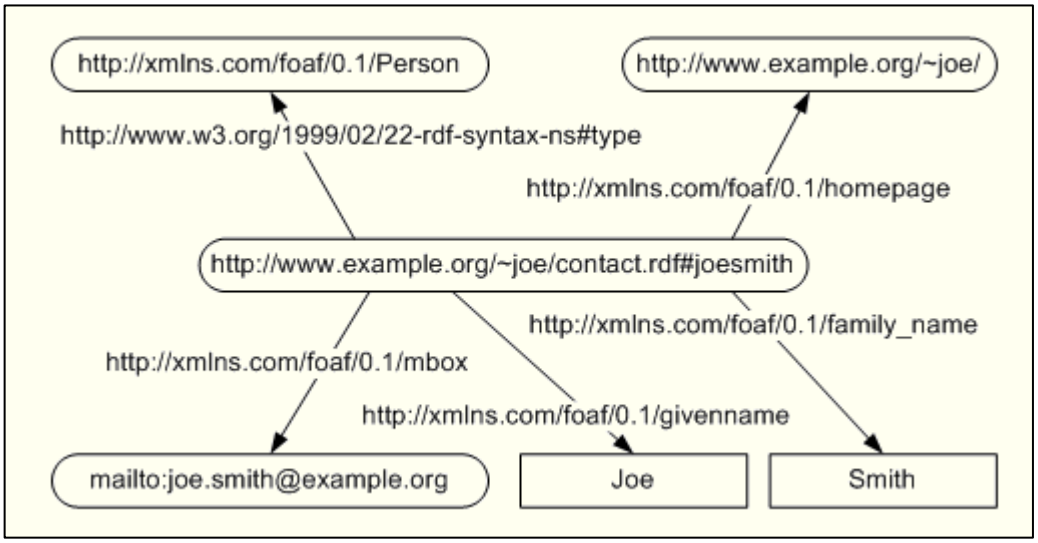
Galvenā GWT ietvara sastāvdaļa ir Java- JavaScript kompilators, kas ģenerē JavaScript kodu. Ģenerētais kods tiek atbalstīts visos populārākajos tīmekļa pārlūkos. Šajā procesā kods ne tikai tiek tulkots no valodas Java uz JavaScript, bet tiek veikta arī koda optimizācija. GWT ģenerētais kods bieži vien ir ātrāks nekā programmētāja rakstītais.

Kompilāciju procesu veic com.google.gwt.dev.GWTCompiler, kurš saņem lietotāja definēto moduli kopā ar konfigurācijas failiem. Kompilēšana tiek sākta no EntryPoint klases. Kompilatora darbības principi ir atšķirīgi no standarta Java kompilatora, jo tas nekompilē visu, kas ir lietotāja definētajā modulī. Tā vietā tiek kompilētas tikai tās daļas, kas tiks lietotas. Kompilators var darboties divos režīmos. Pirmais režīms ir "obfuscate". Tas ģenerē cilvēkam nelasāmu JavaScript kodu, kas tiek darīts ne tik daudz, lai citi nevarētu šo kodu izmantot, bet gan ar nolūku programmu padarīt ātrāku. Otrs ir tā sauktais "pretty" režīms – šajā gadījumā tiek ģenerēts lasāms, strukturēts kods. [6]

GWT piedāvā arī plašu bibliotēku ar komponentēm un paneļiem, tādējādi, iespējams, viegli būvēt tīmekļa lietotnes, kas ir līdzīgas standarta darbvirsmas lietotnēm. Komponenšu bibliotēka ietver visus tipiski lietotās komponentes – teksta ievadlauki, izkrītošā izvēlne, radio pogas, ķekšķastes un citas komponentes. Ir iekļautas arī saliktās komponentes – izvēlnes rīkjosla, koki, dialoga logi un citas.

4.2. RDF

RDF(„Resource Description Framework") ir W3C standarts informācijas aprakstīšanai Semantiskajā tīmeklī. RDF ir datu modelis, kurā dati ir izteikti subjekts-predikāts-objekts trijnieku veidā. Katru šādu trijnieku sauc par apgalvojumu. Kombinējot vairākus apgalvojumus, tiek iegūts RDF datu grafs. [3]



4.1. att. **RDF grafa piemērs**

Šajā piemērā ir redzama informācija par personu vārdā Joe Smith. Attēlā redzami pieci apgalvojumi:

- Resurss ***http://www.example.org/~joe/contact.rdf#joesmith*** ir persona
- Šī resursa e-pasta adrese ir joe.smith@example.org
- Resursa vārds ir Joe
- Resursa uzvārds ir Smith
- Resursa mājaslapa ir http://www.example.org/~joe/

Attēlā redzamajā RDF grafa reprezentācijā ar taisnstūra objektiem tiek apzīmētas literālas vērtības, bet noapaļotajos objektos ir resursi, kuri arī var atsaukties uz citiem resursiem vai uz literārām vērtībām. Bultas apzīmē predikātus.

RDF pasaulē katram resursam (gan subjektiem, gan predikātiem) tiek piešķirts savs unikāls identifikators, URI(„Uniform resource identifier"). URI var būt jebkāda simbolu virkne. Šādi piešķirot katram resursam, savu unikālu identifikatoru ir iespējams ērti atsaukties uz citiem resursiem.

4.1.1. RDF datu aprakstīšanas formāti

RDF dati var tikt glabāti vairākos formātos:

Pirmais un biežāk lietotais formāts ir RDF datus aprakstīt XML formātā (RDF/XML). Šeit neliels piemērs kas apraksta iepriekš definēto resursu:

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:foaf="http://xmlns.com/foaf/0.1/"
    xmlns="http://www.example.org/~joe/contact.rdf#">
  <foaf:Person rdf:about=
        „http://www.example.org/~joe/contact.rdf#joesmith">
    <foaf:mbox rdf:resource="mailto:joe.smith@example.org"/>
    <foaf:homepage
          rdf:resource="http://www.example.org/~joe/"/>
    <foaf:family_name>Smith</foaf:family_name>
    <foaf:givenname>Joe</foaf:givenname>
  </foaf:Person>
</rdf:RDF>
```

Otrs biežāk lietotais formāts ir N3. Attēlā redzams kā tie paši dati, kas redzami iepriekšējā fragmentā, tiek aprakstīti N3 formātā.

```
@prefix :     <http://www.example.org/~joe/contact.rdf#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

:joesmith a foaf:Person ;
      foaf:givenname „Joe" ;
      foaf:family_name „Smith" ;
      foaf:homepage <http://www.example.org/~joe/> ;
      foaf:mbox <mailto:joe.smith@example.org> .
```

Eksistē vēl arī citi formāti, kā piemēram – Turtle un N-Triples, bet šie formāti nav tik populāri kā RDF/XML un N3.

4.1.2. RDFs

RDFs(„RDF schema”) ir RDF vārdnīca, kura ļauj aprakstīt klašu un relāciju taksonomijas. RDFs var uzskatīt par shēmu, kura apraksta kāda veida attiecības var būt starp datiem.

RDFs ievieš virsklases/apakšklases attiecības klasēm. Asociācijām var tikt piekārtota sākuma gala klase(”domain class”) un beigu gala klase(”range class”). Ar RDFs resursu rdf:type var norādīt resursa piederību klasei.

4.1.3. SPARQL

SPARQL ir pieprasījumu valoda, ar kuras palīdzību var pieprasīt un rediģēt informāciju no datiem, kuri ir aprakstīti RDF formātā. SPARQL ir oficiālā W3C rekomendācija, un tiek uzskatīta par vienu no semantisko tehnoloģiju pamatiem. [5]

Kā rezultāts SPARQL vaicājumam tiek atgriezts trijnieks vai arī atbilstošais grafa gabals.

SPARQL vaicājuma piemērs:

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE { ?x foaf:name ?name .
?x foaf:mbox ?mbox . }
```

Šis pieprasījums atgriež visu personu vārdus un e-pasta adreses, kuras ir atrodamas datubāzē.

4.1.4. RDF datu glabāšana

Informāciju, kas aprakstīta RDF formātā ir nepieciešams kaut kur uzglabāt. Tam ir vairāki varianti. Viens no iespējamajiem variantiem būtu glabāt RDF datus .rdf faila veidā kādā no iepriekš minētajām sintaksēm, tomēr šim risinājumam ir dažas nepilnības. Piemēram, ja grafā ir vairāki miljoni trijnieku, datu pieprasīšana un rakstīšana būs ļoti lēna, kā arī pastāvēs datu drošības apdraudējums.

Otrs variants būtu glabāt RDF datus parastā relāciju datubāzē, kas nodrošinātu gan datu drošību, gan ātrdarbību, tomēr paliek jautājums, kā datus pieprasīt. Dabiskais ceļš RDF datu atlasīšanai būtu lietot SPARQL, tādēļ būtu nepieciešama sava veida pieprasījumu translācija.

Trešais un arī šeit lietotais risinājums ir izmantot Virtuoso universālo serveri. Virtuoso ir starpprogrammatūra un datubāzes dzinis, kas atbalsta RDBMS, ORDBMS, RDF un XML funkcionalitātes. Virtuoso atbalsta RDF datu glabāšanu, kā arī to pieprasīšanu ar SPARQL vaicājumu valodu loģiskā līmenī un atbalsta datubāzu draiveru sistēmu, kas ļauj pieslēgties datubāzei no programmatūras. Tiek nodrošināts arī tas, ka dati tiek kārtoti un šādā veidā datubāze strādās ātri arī pie liela apjoma datiem. [4]

4.2. Arhitektūra

Platformas arhitektūras svarīgākās komponentes ir formu definīcijas metamodelis, izpildes laika metamodelis, skriptu valoda un funkciju bibliotēka.

Ar definīcijas metamodeļa palīdzību lietotājs definē lietotnes saskarni. Kā rezultāts tiek iegūta formu definīcijas metamodeļa instanču diagramma, kura ir aprakstīta RDF formātā, un tiek glabāta datubāzē.

Izpildes laika metamodelis ir realizētas ar Java klasēm, un kalpo par platformas realizācijas pamatu. Izpildes laika klases instances tiek veidotas, par pamatu ņemot lietotāja definēto konfigurācijas informāciju. Izpildes laika metamodeļa prezentācijas klašu instances tiks saistītas ar atbilstošajām GWT komponenšu klases instancēm.

4.2.1. Formu definēšanas metamodelis

Formu definēšanas metamodelis paredzēts, lai aprakstītu lietotāja izveidotās saskarnes dizainu un darbības principus. Kā saskarnes aprakstīšanas rezultāts būs RDF grafs, kurš atbildīs formu definēšanas metamodeļa instanču diagrammai. Izveidotais RDF grafs tiks glabāts RDF datubāzē.

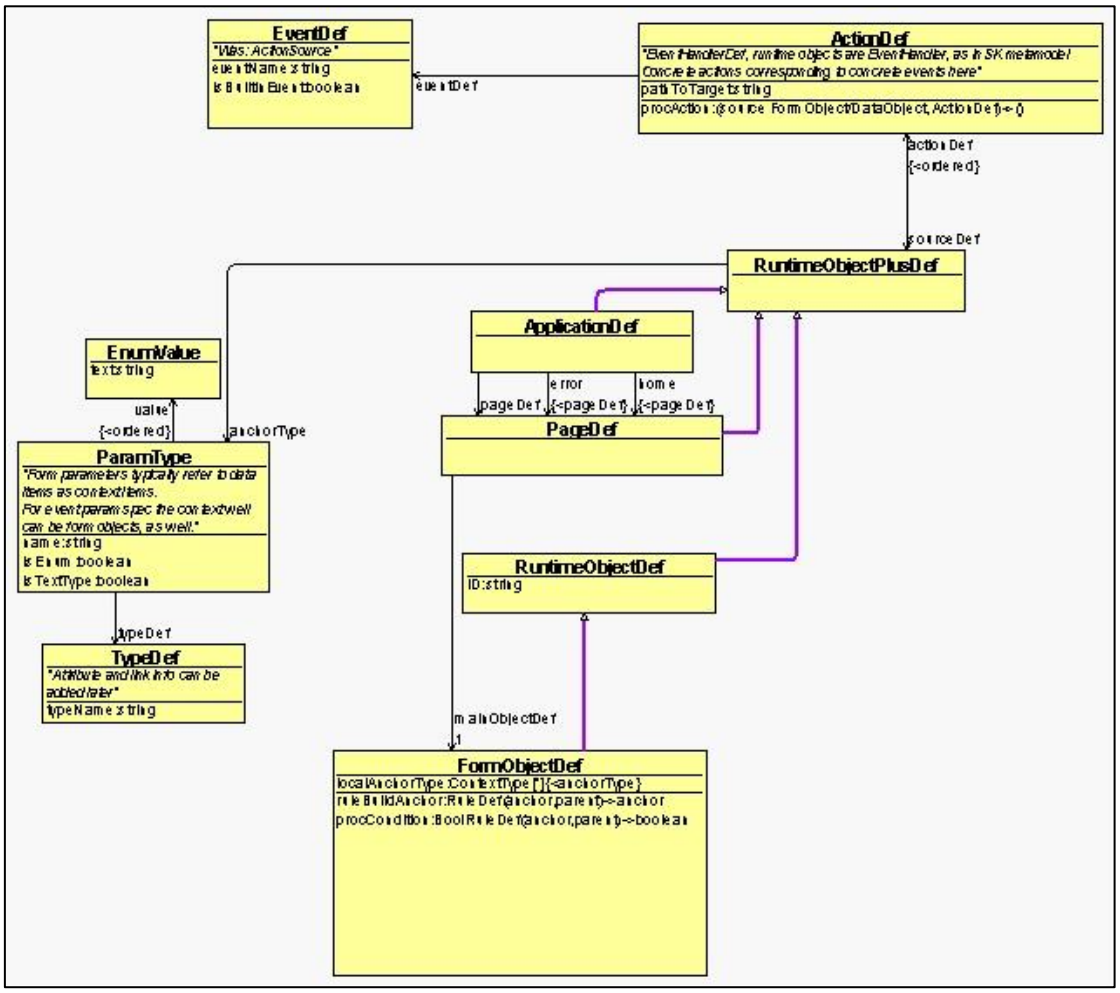
Lietotājam būs pieejami dažādi šabloni, kurus lietotājs varēs izmantot par pamatu savas sistēmas būvēšanai.

Pilns formu definēšanas metamodelis pieejams pielikumā.

4.2.1.1. Formu definēšanas metamodeļa klases

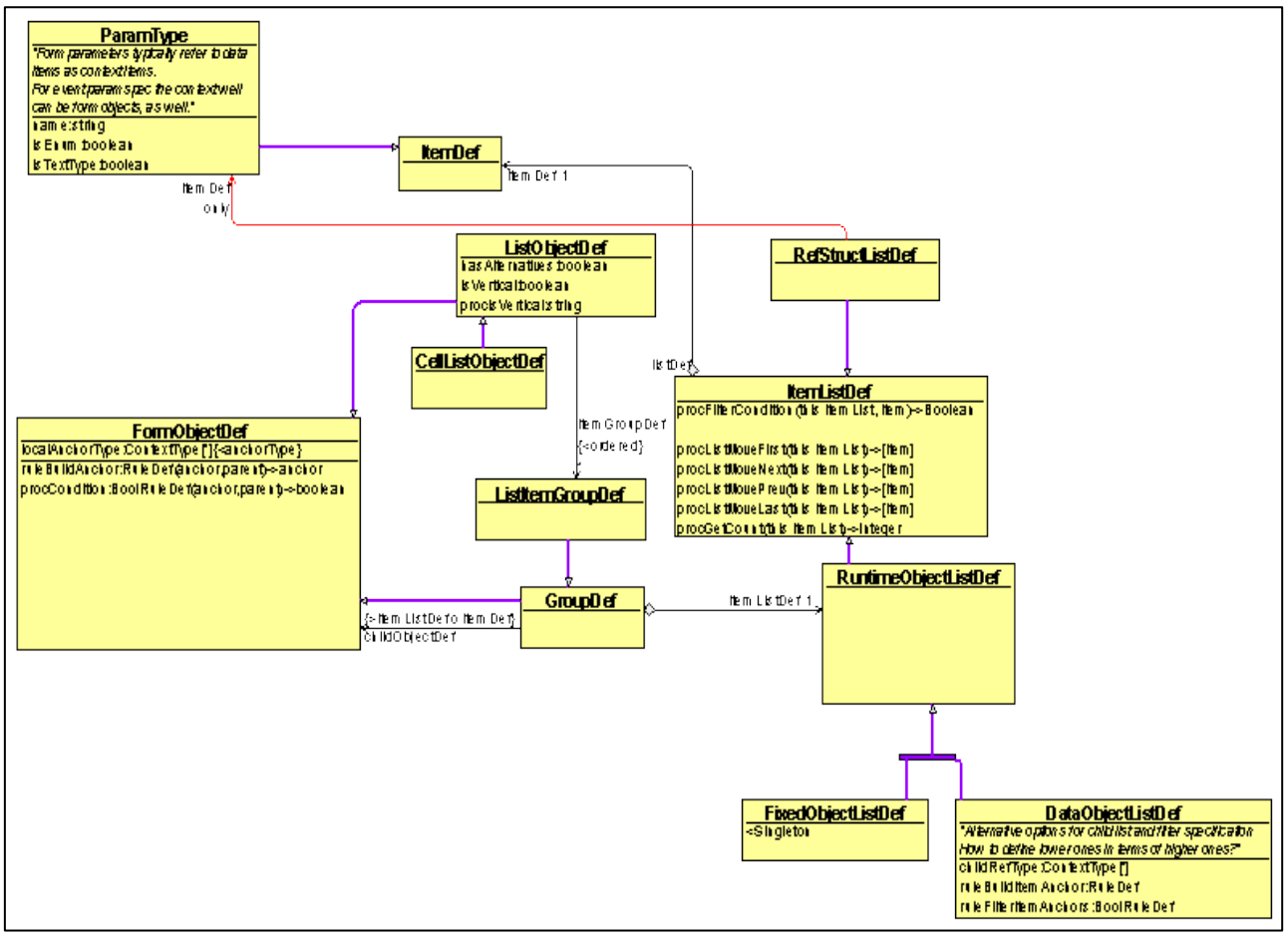
Attēlā 4.2. parādīts fragments no formu definēšanas metamodeļa. Lietotne(*Application*) sastāv no vienas vai vairākām tīmekļa lapām. Katrā lapā ir viens galvenais formas objekts. Katrai metamodelī definētajai instancei var tikt pievienots konteksts. Kontekstam ir nosaukums un pazīme, vai tā tips ir simbolu virkne.

Modeļa instancēm var definēt noteiktas darbības. Kādam no formas objektiem, teiksim, teksta šūnai var pievienot darbību, kad uz tās tiek nospiests, tās teksts pazūd. Darbības veids tiek definēts ar *eventName*, bet pati darbība ar *procAction*.



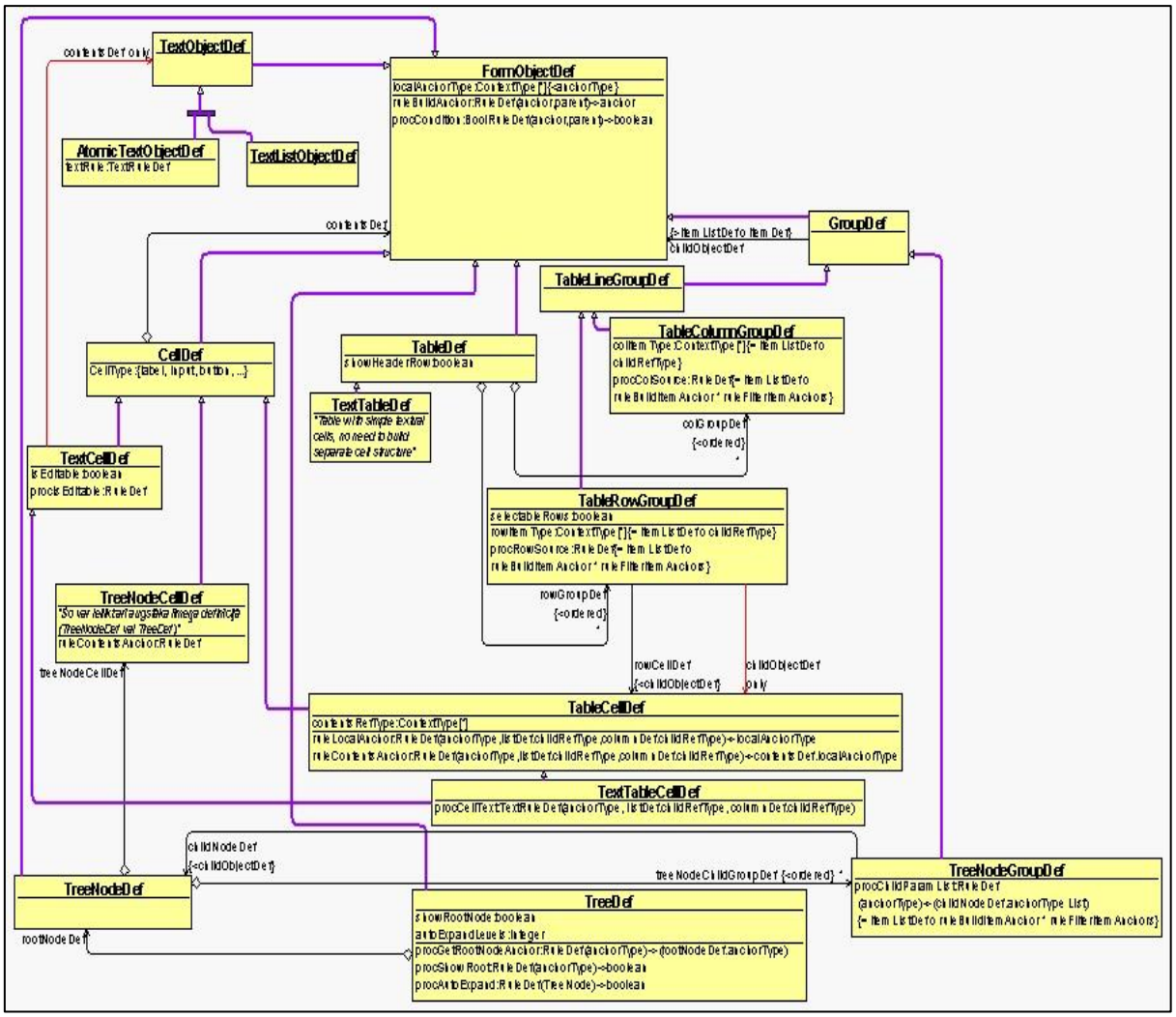
4.2. att. **Formu definēšanas metamodeļa fragments**

Attēlā 4.3. parādīts metamodeļa fragments, kurā attēlota kontekstu saraksta struktūra. Klases *ItemListDef* instances sastāv no *ParamType* klases instancēm (asociācija *itemDef*). Tāpat klase satur metodes, kas ļauj apstaigāt kontekstu sarakstus. Pašu sarakstu būvēšana tiek aprakstīta ar klases *DataObjectListDef* metodi *ruleBuildItemAnchor*, kuras atribūta vērtība ir lietotāja definēts skripts un kurš apraksta kādā veidā tiek iegūtas saraksta vērtības. Skripts tiks interpretēts un izveidots izpildes daļā.



4.3. att. **Formu definēšanas metamodeļa fragments**

Šajā metamodeļa fragmenta parādīta formas objektu definēšanas struktūra (attēls 4.4.). Ar metamodeli iespējams definēt dažāda veida šūnu grupu struktūras – sarakstus, tabulas un kokus. Modeļa priekšrocība ir tāda, ka, piemēram, tabulas šūna var saturēt ne tikai tekstuālas vērtības, bet arī citas formu struktūras, piemēram, sarakstu vai koku. Koks mezgli var saturēt tabulas. Katras datu struktūras modelis ir aprakstīts samērā smalki ar mērķi ļaut veidot sarežģītas struktūras.



4.4. att. **Formu definēšanas metamodeļa fragments**

4.3.2. Izpildes laika metamodelis

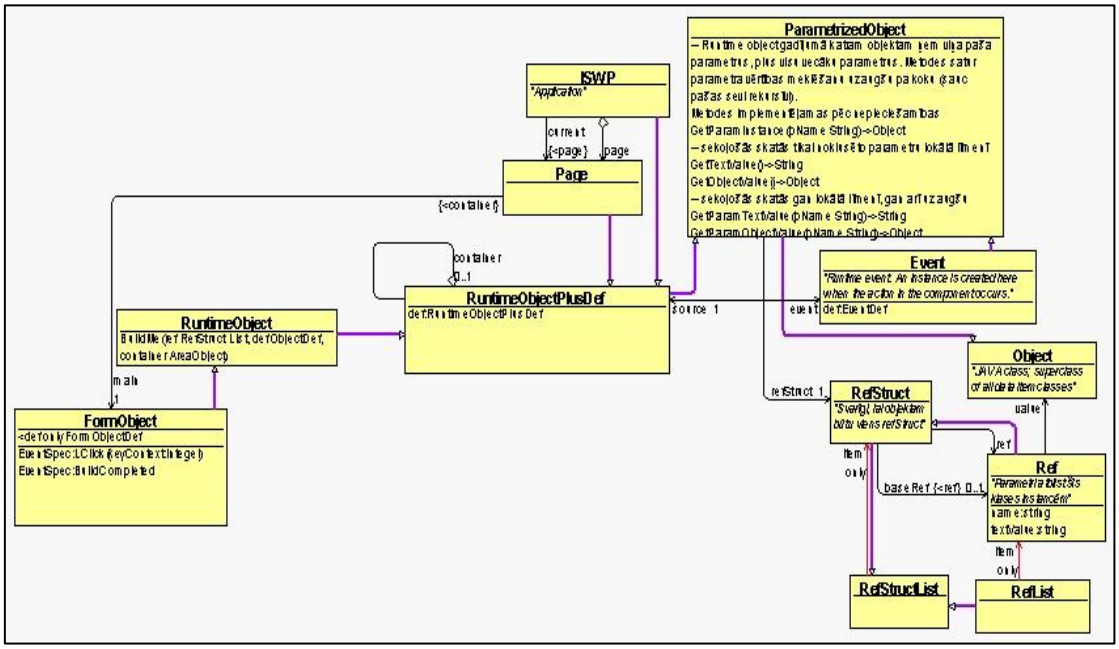
Izpildes laika metamodelis ir līdzīgs formu definēšanas metamodelim, tomēr tie ir paredzēti dažādiem mērķiem un starp tiem ir atšķirības. Izpildes laika metamodeļa klases tiks implementētas Java klašu veidā, savukārt formu definēšanas metamodelis, kā tika aprakstīts iepriekšējā nodaļā, paredzēts lietotnes struktūras definēšanai.

Izpildes modeļa klases instances tiek veidotas lasot konfigurācijas instanču diagrammu, kura atrodas datubāzē RDF datu veidā. Kad izpildes modeļa klašu struktūra ir izveidota, tā tiek saistīta ar GWT prezentācijas daļu.

Pilns izpildes laika metamodelis pieejams pielikumā.

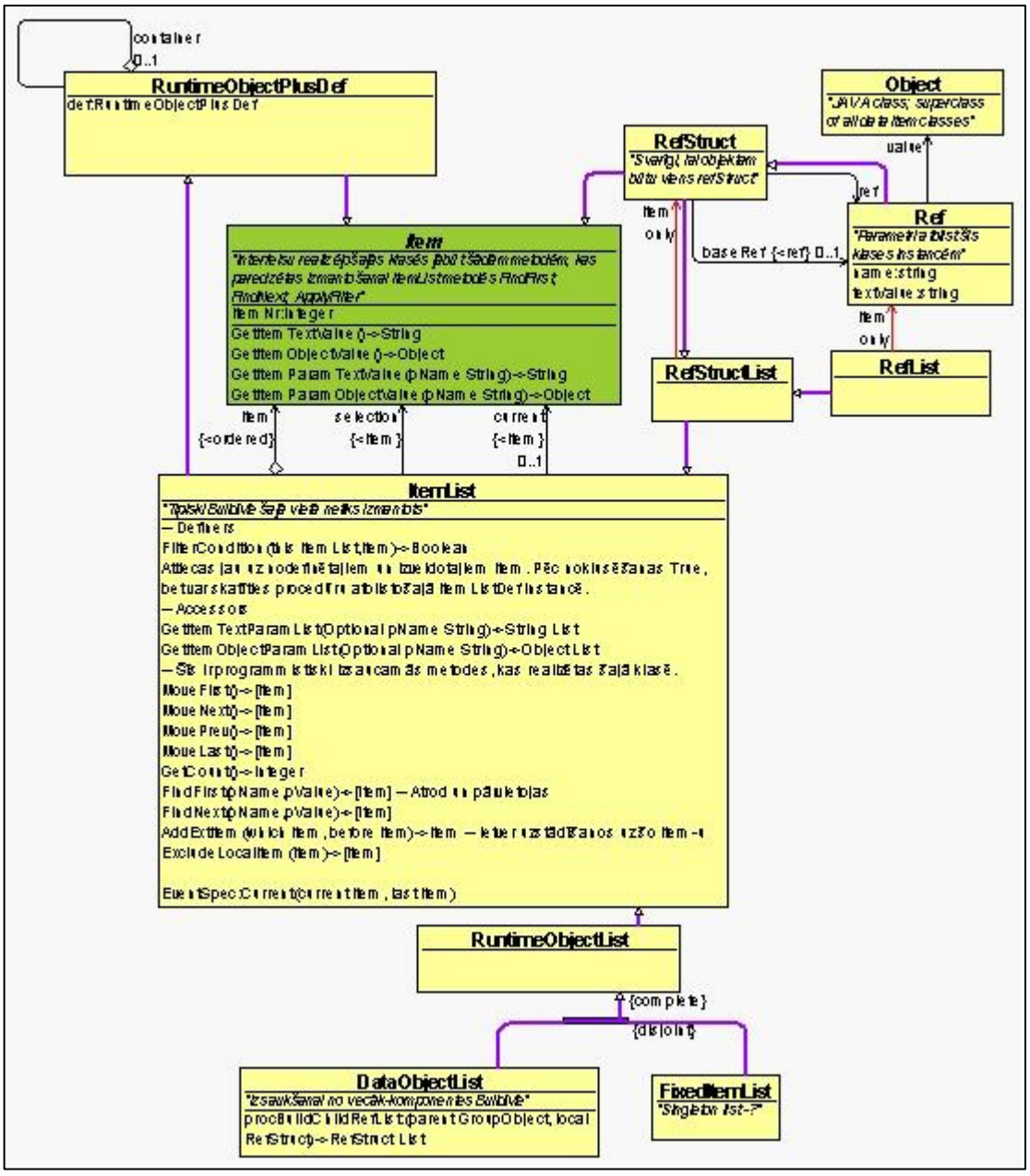
4.3.2.1. Izpildes laika metamodeļa klases

Šajā metamodelī, tāpat kā formu definēšanas modelī, katrai klases instancei var definēt parametru struktūru. Katrai instancei atbildīs viens *refStruct,* kuram būs bāzes parametrs un parametru saraksts. Klase ISWP definē tīmekļa aplikāciju, kura sastāv no vairākām lapām, bet RuntimeObjectPlus pievieno instancēm tās definīcijas datus. Viena no svarīgākajām metodēm ir metode BuildMe, kura definēta klasē RuntimeObject. Šī metode darbojas kā konstruktors - saņem vecāku parametrus un arī pašai definētos parametrus un izveido doto instanci.



4.5. att. **Izpildes laika metamodeļa fragments**

Šajā modeļa fragmentā attēlota parametru sarakstu struktūra. Saraksts kā saraksta elementu satur RefStruct instances (RefStruct implementē Item interfeisu). Sarakstā pieejamas metodes, kas ļauj pa to pārvietoties un meklēt parametru pēc vārda. Tālāk no parametru saraksts saraksta var tikt veidoti dažādi formas elementi, kuri attēlos saraksta vērtības, piemēram, koks ar ontoloģijas klasēm.



4.6. att. **Izpildes laika metamodeļa fragments**

4.3.2. Skriptu valoda un funkciju bibliotēka

Projekta ietvaros tika izstrādāta vienkārša valoda, ar kuras palīdzību lietotājs var definēt likumus, kuri nodrošina sasaisti starp dažādām lapas komponentēm, kā arī starp komponentes un tās bērnu attēlojumiem. Valodas gramatika ir attēlota zemāk esošajā koda fragmentā.

```
Script      :     rule (';' rule)* ;
rule  :     ((varList)? ('<-'|'<<-'))? Term;
term  :     iterm ('++' iterm)*;
iterm :     (navigLink '.')* task Distinct?;
navigLink : (navigText | varspec | filter) Distinct?;
filter      :     '[' term '=' term ']';
task  :     attrib | varspec | functionName? '(' termList? ')' | qString |
number | Bool;
termList:   term (',' term)*;
varList     :     varspec (',' varspec)*;
varspec     :     '@' varName;
navigText : ID;
varName     :     ID;
attrib      :     ID;
functionName :    ID;
qString     :     STRING;
number      :     INT;
Distinct:   '!';
Bool  :     'true' | 'false';
STRING :  '"' ( ~('"') )* '"' ;
ID  : ('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'0'..'9'|'_')*
INT : '0'..'9'+;
WS  :   ( ' '| '\t'| '\r'| '\n') {$channel=HIDDEN;};
```

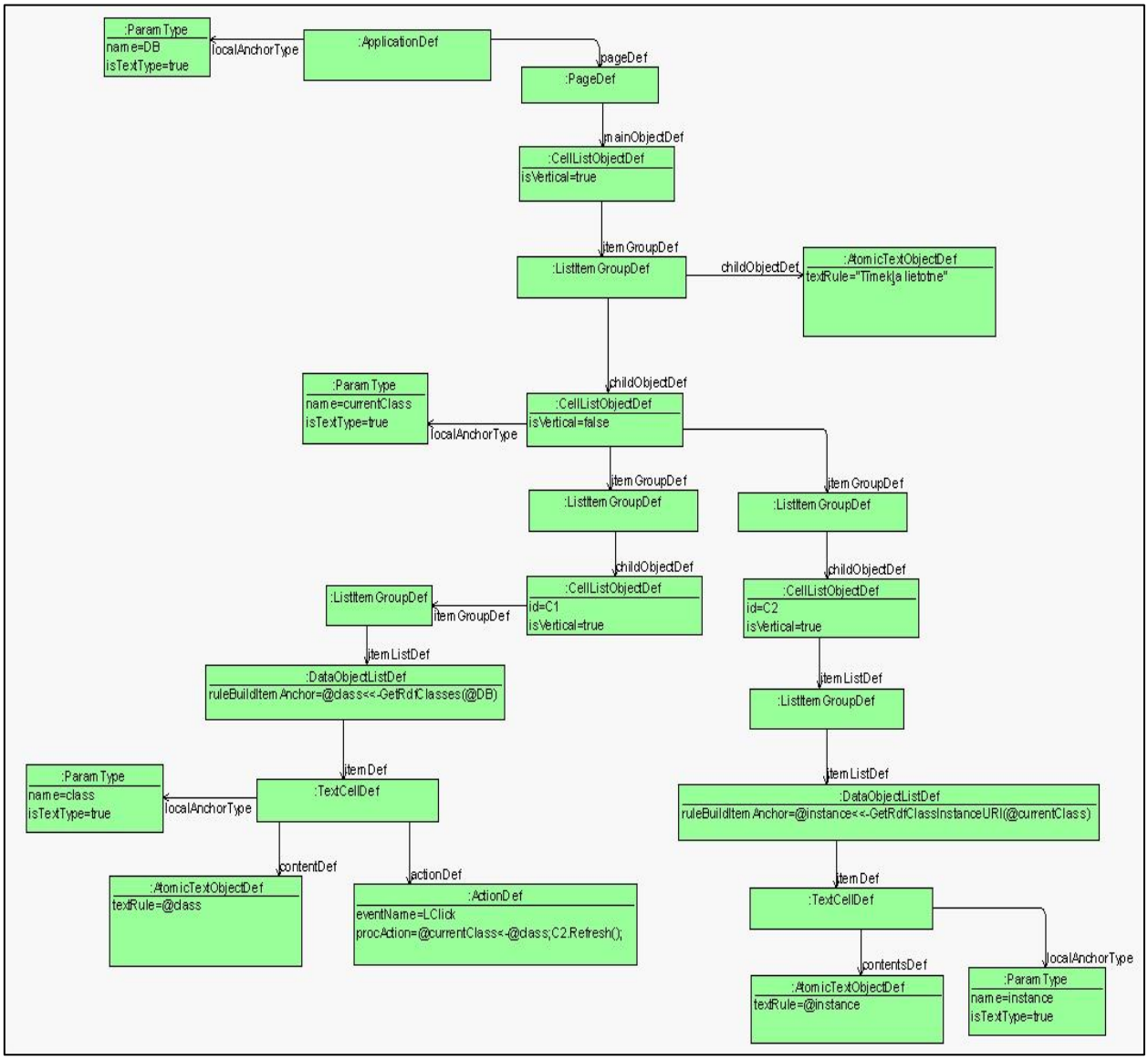
Piemēram, skripts *@class<-GetRDFClasses(@db);* definē mainīgo class, kurā tiek ierakstītas vērtības, kas tiek atgrieztas no funkcijas GetRDFClasses un kurai kā parametrs tiek padots mainīgais db. Funkcija GetRDFClasses ir pieejama funkciju bibliotēkā, tāpat kā visas lietotājam pieejamās funkcijas.

Šobrīd valodai tiek izstrādāts uz abstraktā sintakses koka apstaigāšanu balstīts interpretators.

4.4. Vienkāršas lietotnes piemērs

Lai uzskatāmāk parādītu arhitektūras darbības principus, šajā nodaļā tiks apskatīts vienkāršas lietotnes piemērs.

Uzdevuma nostādne ir šāda – izveidot lietotni ar vienu lapu. Lapa saturēs lietotnes nosaukumu un tajā tiks attēloti divi saraksti. Pirmais saraksts saturēs visas datubāzē esošās klases. Otrs saraksts saturēs visas tās klases instances, kura tiks izvēlēta no pirmā saraksta. Datu ontoloģija un dati glabāsies RDF datubāzē.



4.7. att. **Instanču diagramma dotajam piemēram**

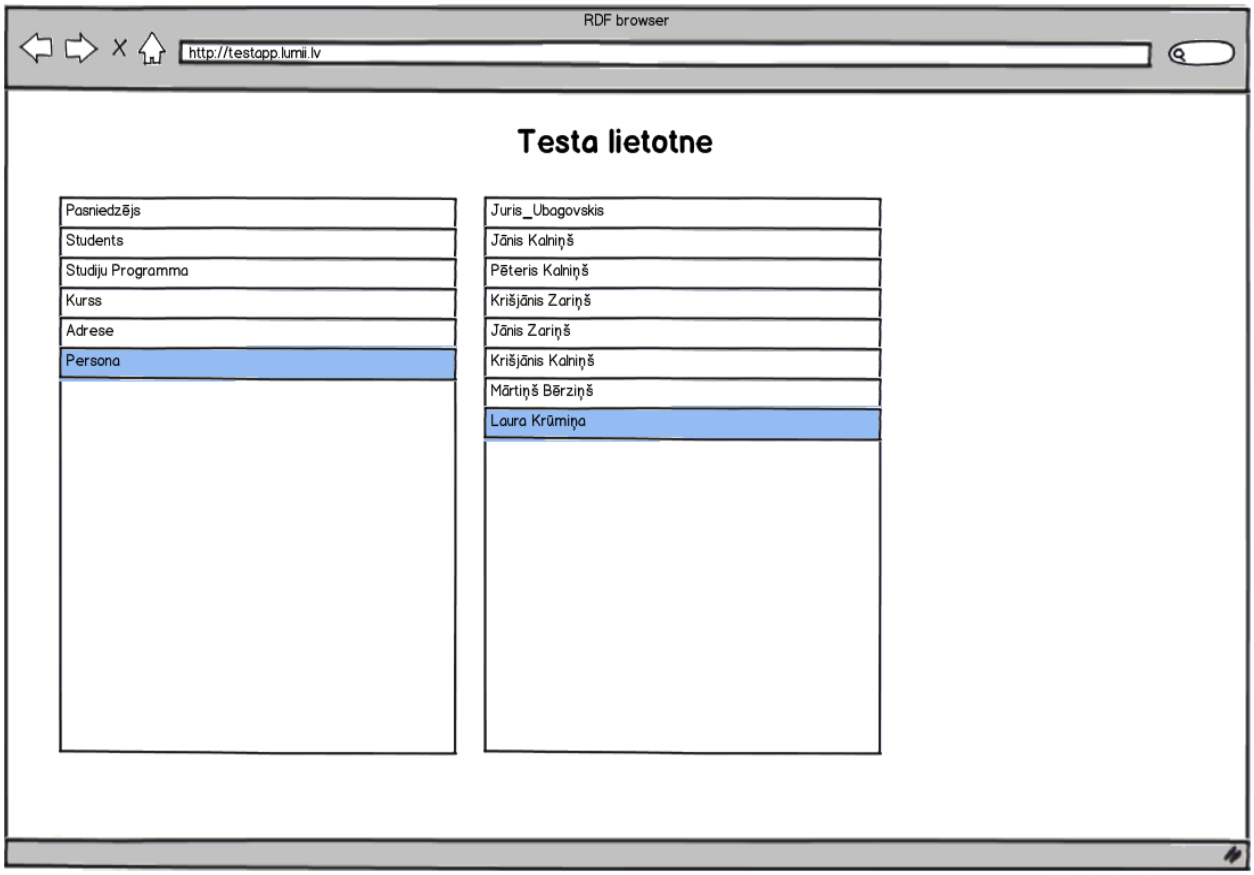
Attēlā redzama lietotnei atbilstošā formu definīcijas metamodeļa instanču diagramma. Iepriekš definēto lietotni varētu aprakstīt arī savādāk, piemēram par pamatu ņemot tabulas struktūru. Šajā gadījumā tiek lietota sarakstu struktūra.

Instanču diagrammā lietotnei (*ApplicationDef*) atbilst parametrs, kurš satur vērtību piekļūšanai pie datubāzes. Lietotnei ir definēta viena lapa, kuras galvenais objekts ir saraksts (*CellListObjectDef*). Šis saraksts sastāv no divām šūnām. Pirmā šūna satur vienkāršu teksta vērtību. Otrā šūna satur vēl vienu sarakstu, kuram ir piekārtots parametrs *currentClass.* Šis parametrs tiek piekārtots šajā vietā, lai abi bērnu saraksti tam varētu piekļūt. Otrās šūnas pirmais saraksts satur datubāzes klašu sarakstu. Saraksts tiek iegūts *DataObjectListDef* instancē ar lietotāja definēta skripta palīdzību – *ruleBuildItemAnchor=@class<<-GetRdfClasses(@DB).* Šis skripts mainīgajā *@class* ieraksta vērtības, kuras atgriež funkcija *GetRDFClasses,* kā parametru saņemot datubāzes nosaukumu.

DataObjectListDef sastāv no teksta šūnām, kurām piekārtots parametrs – viens no klašu saraksta elementiem. Teksta šūna satur teksta objektu, kura vērtība tiek iegūta no mainīgā *@class.*

Teksta šūna satur arī darbības definīciju. Šī darbība definē to, ka nospiežot uz kādas klases pirmajā sarakstā, tiks atlasītas tās instances. Šīs instances tiks attēlotas otrajā sarakstā. Redzams, ka darbības loģika ir nodefinēta ar vienkārša skripta palīdzību - *procAction=@currentClass<-@class;C2.Refresh().* Skripta pirmā daļa kā aktīvo klasi (*currentClass)* saglabā to klasi, kura tika izvēlēta. Otrā daļa – *C2.Refresh();* atjauno C2 komponenti. Instanču diagrammā redzams, ka C2 ir saraksts, kurā tiek rādītas no pirmā saraksta izvēlētās klases instances. Otrā saraksts izveides darbības principi ir līdzīgi kā pirmā saraksta (klašu saraksta) izveidei. Kā saraksta elementu iegūšana ir definēts skripts *ruleBuildItemAnchor=@instance<<-GetRdfClassInstanceURI(@currentClass).* Tiek izveidots saraksts ar instancēm, *@instance.* Sarakstu atgriež funkcija *GetRdfClassInstanceURI,* kura kā parametru saņem mainīgo ar klases nosaukumu. Šī funkcija ir pieejama funkciju bibliotēkā. Tāpat kā pirmajā sarakstā, *DataObjectListDef* sastāv no teksta šūnām, kuru satur teksta objektu. Objekta vērtība tiek iegūta no parametra *@instance.*

No instanču diagrammas tiek ģenerēta lietotne, kuras attēls izskatītos šādi:



4.8. att. **Lietotnes piemērs**

SECINĀJUMI

Eksistē vairākas tehnoloģijas, kuras ļauj lietotājam veidot saskarnes darbam ar datiem bez vajadzības programmēt. Tomēr tām visām eksistē savi trūkumi, tāpēc šajā nišā ir brīva vieta jaunām tehnoloģijām.

Darbā aprakstītajai arhitektūrai, salīdzinot ar apskatītajiem piemēriem, ir vairākas priekšrocības. Pirmkārt tiek piedāvātas brīvākas formas struktūras definēšanas iespējas, kā arī ir iespējams attēlot dažādām struktūrām atbilstošus datus. Otrkārt saskarnes konfigurāciju iespējams veidot kā metamodeļa instanci un glabāt to RDF datubāzē.

Ar šādu platformu ātri varētu izstrādāt vienkāršas tīmekļa lietotnes darbam ar RDF datiem. Darba autors izstrādāja pirmajā nodaļā aprakstīto RDF pārlūku trīs mēnešu laikā, veicot ievērojamu programmēšanas darbu. Ar šīs platformas palīdzību izveidot pārlūku ar šādu funkcionalitāti būtu iespējams daudz ātrāk. Tādējādi varam redzēt ieguvumus, ko sniegtu šāda platforma.

PATEICĪBA

Darba autors izsaka pateicība darba vadītājam Kārlim Čerānam par palīdzību bakalaura darba izstrādē.

IZMANTOTĀ LITERATŪRA UN AVOTI

Raksti

1. **Arnicāns, G., Karnītis, Ģ.** Heterogeneous Database Browsing in WWW Based on Meta Model of Data Sources, 2001.

Disertācijas, maģistru un bakalauru darbi

2. **Romāne, A.** Semantisko tehnoloģiju loma tīmekļa lietojumprogrammu izstrādē. LU Datorikas fakultāte. Rīga : Latvijas Universitāte, 2011.

Elektroniskie informācijas avoti

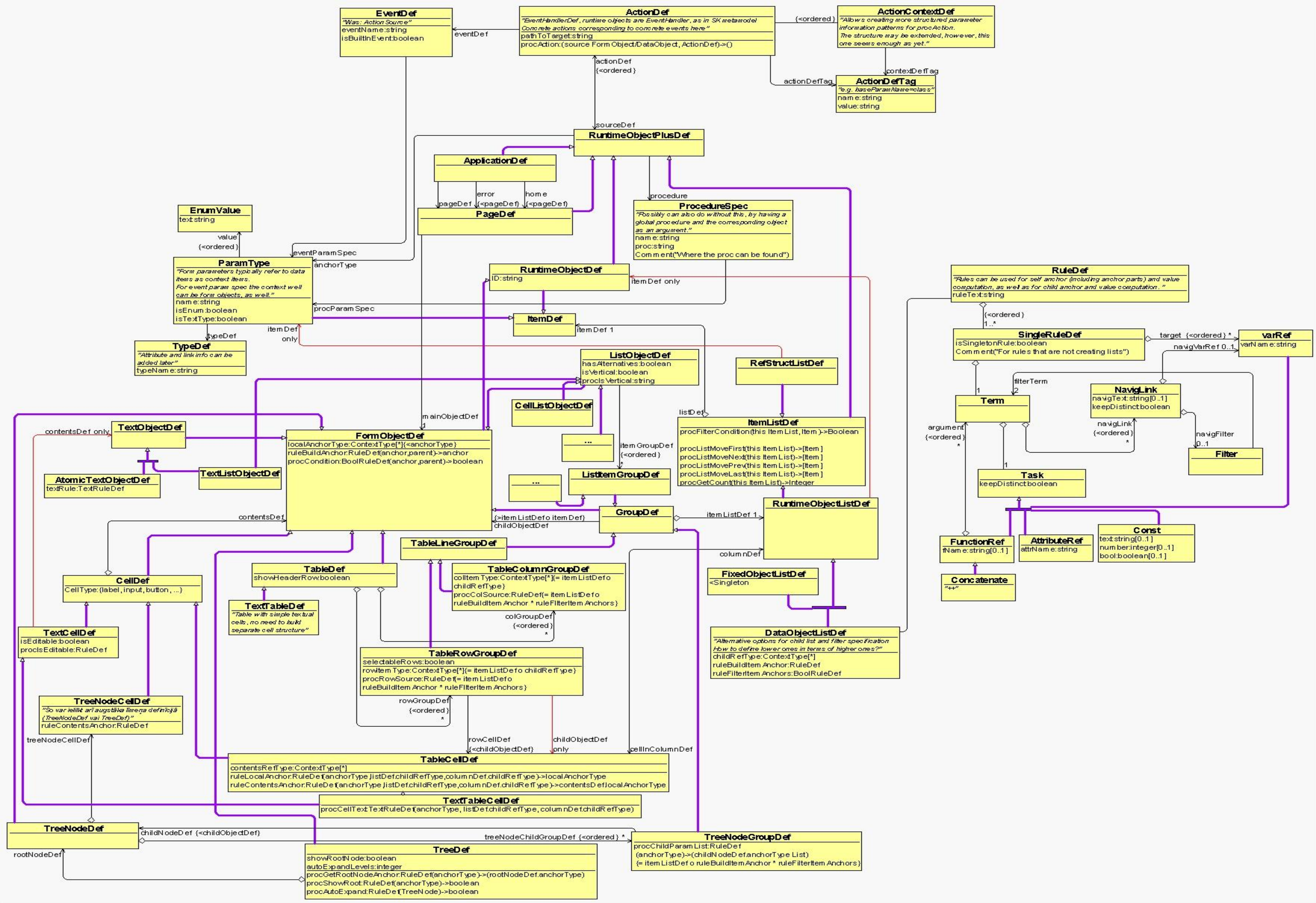
3. **Manola, F., Miller, E.** RDF Primer. [tiešsaiste] – [atsauce 23.05.2012.]. Pieejams: http://www.w3.org/TR/rdf-syntax/

4. OpenlinkVirtoso Universal server. [tiešsaiste] – [atsauce 25.05.2012.]. Pieejams: http://docs.openlinksw.com/virtuoso/

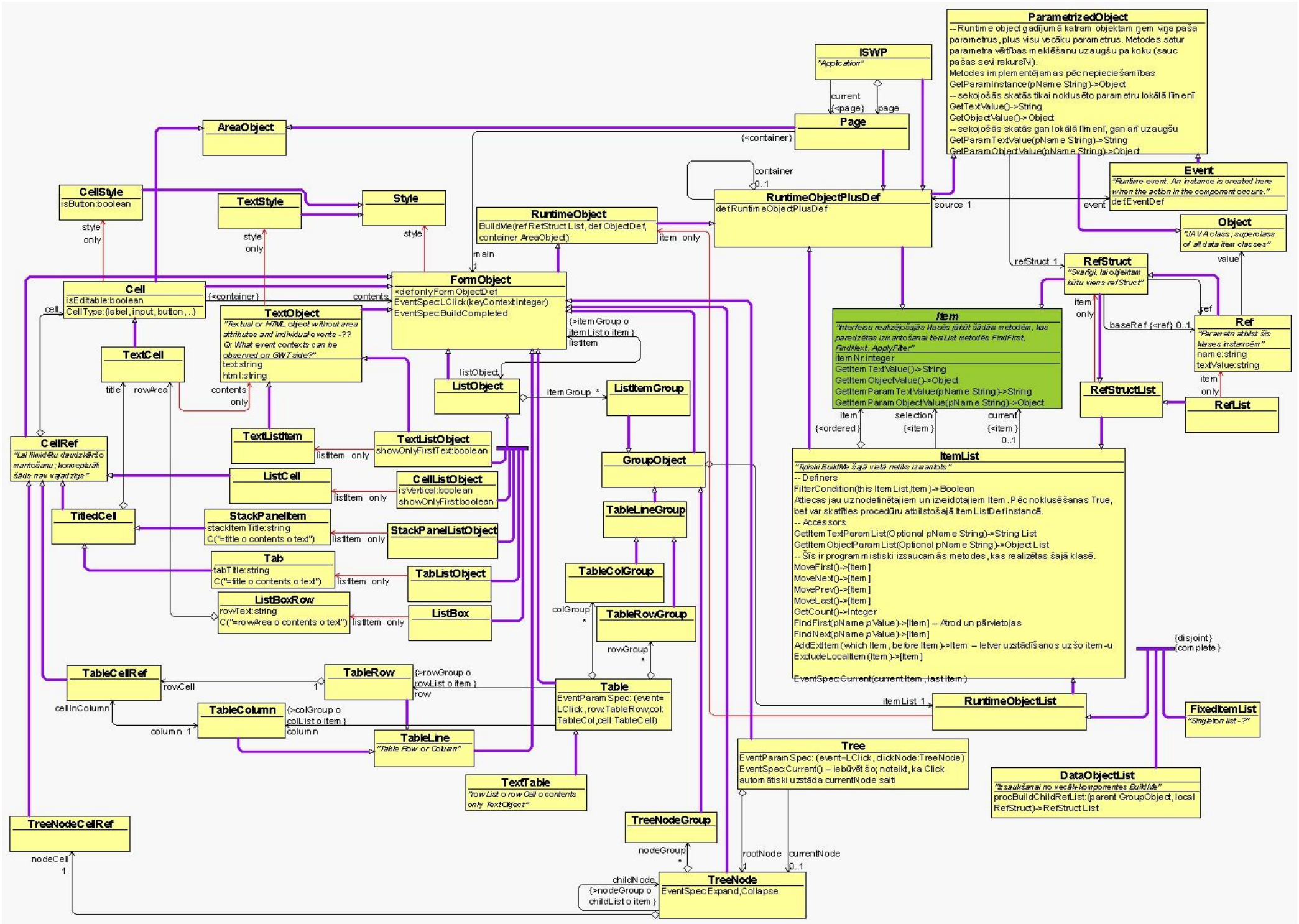
5. SPARQL Query Language for RDF. [tiešsaiste] – [atsauce 25.05.2012.]. Pieejams: http://www.w3.org/TR/rdf-sparql-query/

6. Google Web Toolkit. [tiešsaiste] – [atsauce 25.05.2012.]. Pieejams: https://developers.google.com/web-toolkit/overview

PIELIKUMI



EventDef
"Was: Action Source"
eventName:string
isBuiltInEvent:boolean
eventDef
ActionDef
"EventHandlerDef, runtime objects are EventHandler, as in SK metamodel
Concrete actions corresponding to concrete events here"
pathToTarget:string
procAction:(source Form Object/DataObject, ActionDef)->()
{<ordered}
ActionContextDef
"Allows creating more structured parameter information patterns for procAction.
The structure may be extended, however, this one seems enough as yet."
contextDefTag
actionDefTag
ActionDefTag
"e.g. baseParamName=class"
name:string
value:string
actionDef
{<ordered}
sourceDef
RuntimeObjectPlusDef
ApplicationDef
pageDef
error
{<pageDef}
home
{<pageDef}
PageDef
procedure
ProcedureSpec
"Possibly can also do without this, by having a global procedure and the corresponding object as an argument."
name:string
proc:string
Comment("Where the proc can be found")
EnumValue
text:string
value
{<ordered}
eventParamSpec
anchorType
ParamType
"Form parameters typically refer to data items as context items.
For event param spec the context well can be form objects, as well."
name:string
isEnum:boolean
isTextType:boolean
procParamSpec
RuntimeObjectDef
ID:string
itemDef only
ItemDef
itemDef 1
typeDef
itemDef
only
TypeDef
"Attribute and link info can be added later"
typeName:string
ListObjectDef
hasAlternatives:boolean
isVertical:boolean
procIsVertical:string
RefStructListDef
CellListObjectDef
mainObjectDef
listDef
ItemListDef
procFilterCondition(this ItemList, Item)->Boolean
procListMoveFirst(this ItemList)->[Item]
procListMoveNext(this ItemList)->[Item]
procListMovePrev(this ItemList)->[Item]
procListMoveLast(this ItemList)->[Item]
procGetCount(this ItemList)->Integer
contentsDef only
TextObjectDef
AtomicTextObjectDef
textRule:TextRuleDef
TextListObjectDef
FormObjectDef
localAnchorType:ContextType[*]{<anchorType}
ruleBuildAnchor:RuleDef(anchor,parent)->anchor
procCondition:BoolRuleDef(anchor,parent)->boolean
...
itemGroupDef
{<ordered}
ListItemGroupDef
...
GroupDef
{>itemListDefo itemDef}
childObjectDef
itemListDef 1
RuntimeObjectListDef
contentsDef
columnDef
TableLineGroupDef
CellDef
CellType:{label, input, button, ...}
TableDef
showHeaderRow:boolean
TableColumnGroupDef
colItemType:ContextType[*]{= itemListDefo childRefType}
procColSource:RuleDef(= itemListDefo ruleBuildItem Anchor * ruleFilterItem Anchors)
TextTableDef
"Table with simple textual cells, no need to build separate cell structure"
colGroupDef
{<ordered}
FixedObjectListDef
<Singleton
DataObjectListDef
"Alternative options for child list and filter specification
How to define lower ones in terms of higher ones?"
childRefType:ContextType[*]
ruleBuildItem Anchor:RuleDef
ruleFilterItem Anchors:BoolRuleDef
TextCellDef
isEditable:boolean
procIsEditable:RuleDef
TableRowGroupDef
selectableRows:boolean
rowItemType:ContextType[*]{= itemListDefo childRefType}
procRowSource:RuleDef(= itemListDefo ruleBuildItem Anchor * ruleFilterItem Anchors)
rowGroupDef
{<ordered}
TreeNodeCellDef
"Šo var ielikt arī augstāka līmeņa definīcijā (TreeNodeDef vai TreeDef)"
ruleContentsAnchor:RuleDef
treeNodeCellDef
rowCellDef
{<childObjectDef}
childObjectDef
only
cellInColumnDef
TableCellDef
contentsRefType:ContextType[*]
ruleLocalAnchor:RuleDef(anchorType,listDef.childRefType,columnDef.childRefType)->localAnchorType
ruleContentsAnchor:RuleDef(anchorType,listDef.childRefType,columnDef.childRefType)->contentsDef.localAnchorType
TextTableCellDef
procCellText:TextRuleDef(anchorType, listDef.childRefType, columnDef.childRefType)
TreeNodeDef
childNodeDef {<childObjectDef}
treeNodeChildGroupDef {<ordered} *
TreeNodeGroupDef
procChildParamList:RuleDef
(anchorType)->(childNodeDef.anchorType List)
(= itemListDefo ruleBuildItemAnchor * ruleFilterItem Anchors)
rootNodeDef
TreeDef
showRootNode:boolean
autoExpandLevels:integer
procGetRootNodeAnchor:RuleDef(anchorType)->(rootNodeDef.anchorType)
procShowRoot:RuleDef(anchorType)->boolean
procAutoExpand:RuleDef(TreeNode)->boolean
RuleDef
"Rules can be used for self anchor (including anchor parts) and value computation, as well as for child anchor and value computation."
ruleText:string
{<ordered}
1..*
SingleRuleDef
isSingletonRule:boolean
Comment("For rules that are not creating lists")
target {<ordered} *
varRef
varName:string
navigVarRef 0..1
filterTerm
NavigLink
navigText:string[0..1]
keepDistinct:boolean
Term
argument
{<ordered}
navigLink
{<ordered}
navigFilter
0..1
Filter
Task
keepDistinct:boolean
FunctionRef
fName:string[0..1]
AttributeRef
attrName:string
Const
text:string[0..1]
number:integer[0..1]
bool:boolean[0..1]
Concatenate
"++"


ParametrizedObject
-- Runtime object gadījumā katram objektam ņem viņa paša parametrus, plus visu vecāku parametrus. Metodes satur parametra vērtības meklēšanu uzaugšu pa koku (sauc pašas sevi rekursīvi).
Metodes implementējamas pēc nepieciešamības
GetParamInstance(pName String)->Object
-- sekojošās skatās tikai noklusēto parametru lokālā līmenī
GetTextValue()->String
GetObjectValue()->Object
-- sekojošās skatās gan lokālā līmenī, gan arī uzaugšu
GetParamTextValue(pName String)->String
GetParamObjectValue(pName String)->Object
ISWP
"Application"
current {<page}
page
Page
AreaObject
{<container}
container 0..1
RuntimeObjectPlusDef
defRuntimeObjectPlusDef
source 1
event
Event
"Runtime event. An instance is created here when the action in the component occurs."
defEventDef
Object
"JAVA class; superclass of all data item classes"
value
CellStyle
isButton:boolean
style only
TextStyle
style only
Style
style
RuntimeObject
BuildMe(ref RefStruct List, def ObjectDef, container AreaObject)
item only
main
refStruct 1
RefStruct
"Svarīgi, lai objektam būtu viens refStruct"
item only
ref
baseRef {<ref} 0..1
Ref
"Parametri atbilst šīs klases instancēm"
name:string
textValue:string
item only
Cell
isEditable:boolean
CellType:{label, input, button, ..}
cell
{<container}
contents
FormObject
<def only Form ObjectDef
EventSpec:LClick(keyContext:integer)
EventSpec:BuildCompleted
{>itemGroup o itemList o item}
listItem
TextObject
"Textual or HTML object without area attributes and individual events -?? Q: What event contexts can be observed on GWT side?"
text:string
html:string
contents only
listObject
Item
"Interfeisu realizējošajās klasēs jābūt šādām metodēm, kas paredzētas izmantošanai itemList metodēs FindFirst, FindNext, ApplyFilter"
itemNr:integer
GetItemTextValue()->String
GetItemObjectValue()->Object
GetItemParamTextValue(pName String)->String
GetItemParamObjectValue(pName String)->Object
RefStructList
RefList
item only
TextCell
title
rowArea
ListObject
itemGroup *
ListItemGroup
item {<ordered}
selection {<item}
current {<item} 0..1
CellRef
"Lai likvidētu daudzkāršo mantošanu; konceptuāli šāds nav vajadzīgs"
TextListItem
listItem only
TextListObject
showOnlyFirstText:boolean
GroupObject
ItemList
"Tipiski BuildMe šajā vietā netiks izmantots"
-- Definers
FilterCondition(this ItemList,Item)->Boolean
Attiecas jau uznodefinētajiem un izveidotajiem Item. Pēc noklusēšanas True, bet var skatīties procedūru atbilstošajā ItemListDef instancē.
-- Accessors
GetItemTextParamList(Optional pName String)->String List
GetItemObjectParamList(Optional pName String)->Object List
-- Šīs ir programmistiski izsaucamās metodes, kas realizētas šajā klasē.
MoveFirst()->[Item]
MoveNext()->[Item]
MovePrev()->[Item]
MoveLast()->[Item]
GetCount()->Integer
FindFirst(pName,pValue)->[Item] – Atrod un pārvietojas
FindNext(pName,pValue)->[Item]
AddExtItem(which Item, before Item)->Item – ļetver uzstādīšanos uzšo item-u
ExcludeLocalItem(Item)->[Item]
EventSpec:Current(currentItem, lastItem)
CellListObject
isVertical:boolean
showOnlyFirst:boolean
ListCell
listItem only
TableLineGroup
TitledCell
StackPanelItem
stackItemTitle:string
C("=title o contents o text")
StackPanelListObject
Tab
tabTitle:string
C("=title o contents o text")
TabListObject
TableColGroup
ListBoxRow
rowText:string
C("=rowArea o contents o text")
ListBox
colGroup *
TableRowGroup
rowGroup *
{disjoint}
{complete}
TableCellRef
rowCell
TableRow
{>rowGroup o rowList o item}
row
Table
EventParamSpec: (event=LClick, row:TableRow, col:TableCol, cell:TableCell)
itemList 1
RuntimeObjectList
FixedItemList
"Singleton list -?"
cellInColumn
TableColumn
{>colGroup o colList o item}
column
column 1
TableLine
"Table Row or Column"
Tree
EventParamSpec: (event=LClick, clickNode:TreeNode)
EventSpec:Current() – iebūvēt šo; noteikt, ka Click automātiski uzstāda currentNode saiti
DataObjectList
"Izsaukšanai no vecāk-komponentes BuildMe"
procBuildChildRefList:(parent GroupObject, local RefStruct)->RefStruct List
TextTable
"row List o row Cell o contents only TextObject"
TreeNodeCellRef
TreeNodeGroup
nodeCell 1
nodeGroup *
rootNode 1
currentNode 0..1
childNode {>nodeGroup o childList o item}
TreeNode
EventSpec:Expand,Collapse